DESIGN AND IMPLEMENTATION OF A REAL-TIME VIDEO-OCULOGRAPHIC GAZE DETECTION AND TRACKING SYSTEM

Cihan TOPAL

PhD Dissertation

Graduate School of Sciences
Computer Engineering Program

August, 2014

This thesis is supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) and Anadolu University Commission of Scientific Research Projects (BAP) under the grant numbers 111E053 and 1207F113, respectively.

ANADOLU ÜNİVERSİTESİ

JÜRİ VE ENSTİTÜ ONAYI

Cihan Topal'ın **Design and Implementation of A Real-time Video-Oculographic Gaze Detection and Tracking System** başlıklı **Bilgisayar Mühendisliği** Anabilim Dalındaki Doktora tezi 24 - 07 - 2014 tarihinde aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim - Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı -Soyadı	İmza
Üye (Tez Danışmanı)	**: Doç. Dr. Cüneyt AKINLAR**	
Üye	**: Prof. Dr. Ömer Nezih GEREK**	
Üye	**: Doç. Dr. Serkan GÜNAL**	
Üye	**: Doç. Dr. Atakan DOĞAN**	
Üye	**: Yard. Doç. Dr. Kemal ÖZKAN**	

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ABSTRACT

PhD Dissertation

DESIGN AND IMPLEMENTATION OF A REAL-TIME VIDEO-OCULOGRAPHIC GAZE DETECTION AND TRACKING SYSTEM

Cihan TOPAL

Anadolu University
Graduate School of Sciences
Computer Engineering Program

Supervisor: Assoc. Prof. Dr. Cuneyt AKINLAR

2014, 116 pages

In this thesis, real-time video-oculographic Point of Gaze (PoG) computation methods have been investigated. A fully functional PoG system generally consists of three main components. The first is to extract features from eye images. The second is to compute PoG based on a mathematical or geometric model using the information obtained in the first step. The third is to design and develop a user-friendly application to operate the computer or allow fast text entry using the accuracy provided by the eye tracking system. To come up with a novel and fully functional eye tracking system at the end of the thesis, research and development for all three components of the system have been performed.

In the first step of the project, a robust and real-time algorithm has been designed to detect the boundary of the pupil in an eye image. The algorithm not only detects the pupil boundary and the center when the pupil is in clear sight, but it also succeeds even in tough occlusive cases where the pupil is partly covered by eye lashes or the eyelid.

In the second step, Point of Gaze has been computed in 3D using the eye features obtained in the first step. The major problem to solve in this step was to compensate head movements during PoG computation. An eye tracking model was developed to support this feature in the proposed system. Using the proposed solution, head movements can be tolerated by calibration and tracking for 3D PoG computation without the need for complex geometric computations.

In the last step, a novel and user-friendly On Screen Keyboard (OSK) has been developed that is used together with the eye tracking system for fast textual entry. In our gaze interface, we try to enhance users' word input rates as they are typing with their glances. Experiments performed with many participants have shown that the developed OSK is easy to adapt and can significantly boost the text input throughput.

Keywords: Eye tracking; Pupil detection; Gaze estimation; Gaze interface; Human-computer interaction; On-Screen Keyboard.

ÖZET

Doktora Tezi

GERÇEK-ZAMANLI VİDEO-OKÜLOGRAFİK BAKIŞ YÖNÜ TESPİT VE TAKİP SİSTEMİNİN TASARIMI VE GERÇEKLEMESİ

Cihan TOPAL

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Cüneyt AKINLAR
2014, 116 sayfa

Bu tez çalışması kapsamında video tabanlı bakış noktası tespit yöntemleri üzerinde çalışılmıştır. Tam anlamıyla işlevsel bir bakış noktası tespit sistemi genel olarak üç ana bileşenden oluşmaktadır. Bunlardan birincisi göz imgelerinden gerekli öznitelikleri çıkarılmasıdır. İkincisi ilk aşamada elde edilen bilgileri kullanarak belirli bir matematiksel ya da geometrik modele göre bakış noktasının tespit edilmesidir. Üçüncüsü ise göz izleme sisteminin sağlayabildiği hassasiyet miktarında bilgisayarı veya başka bir cihazı kullanabilmeyi sağlayan arayüz uygulamalarıdır.

Tez çalışmaları süresince özgün ve tamamıyla işlevsel bir göz takip sistemi geliştirilmesi amaçlanmış ve bu doğrultuda göz takip sistemini olusturan üç bileşen için de araştırma ve geliştirme faaliyetleri gerçekleştirilmiştir. İlk aşama için göz imgelerinden gürbüz ve hızlı bir şekilde gözbebeği konturunu çıkaran bir algoritma tasarlanmıştır. Bu algoritmalardan beklenen en önemli özellikler göz kapağı, kirpikler ve yansımalardan dolayı oluşan kapatmaların olduğu durumlarda bile çalışabilmeleri ve gözbebeğinin merkezini hassas bir şekilde tespit edebilmeleridir. Geliştirilen gözbebeği tespit algoritmasında bu problemlerin çözümü amaçlanmış ve önemli ölçüde ilerleme sağlanmıştır.

İkinci aşamada göz imgelerinden elde edilen öznitelikler kullanılarak bakış noktasının 3-boyutta tespit edilmesi amaçlanmıştır. Bu adımda çözülmesi gereken en önemli problem geliştirilen yöntemin baş hareketlerini telafi ederek gürbüz bir şekilde çalışabilmesidir. Önerilen sistemle bu özelliği destekleyen bir göz izleme modeli geliştirilmiştir. Bu modelle hem kalibrasyon hem de izleme aşamalarında gerçekleşen baş hareketlerinin telafisi önemli ölçüde sağlanmakta, hem de diğer 3B bakış noktası tespit yöntemleri gibi zahmetli geometrik kalibrasyon gerektirmemektedir.

Son aşamada ise göz izleme sistemleriyle kullanılırken hızlı bir sekilde metin girişi sağlayabilecek özgün bir klavye arayüzü tasarlanmış ve geliştirilmiştir. Birçok farklı kullanıcıyla gerçekleştirilen deneylerin sonucunda geliştirdiğimiz arayüzün son derece başarılı olduğu ispatlanmıştır.

Anahtar Kelimeler: Göz izleme; Gözbebeği tespiti; Bakış noktası tespiti; İnsan-bilgisayar etkileşimi; Ekran klavyesi.

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Cüneyt Akınlar for his support and guidance during my graduate studies. Dr. Akınlar has always provided his time to supervise my research objectives, at the same time, allowed me the freedom to pursue my own prospective solutions, which is greatly appreciated. I would also thank Burak Benligiray who is the other member of our team for his effort on developing various parts of the thesis project.

I would like to acknowledge the members of the thesis committee because of their participation and contributions in the thesis process. I would also thank to Dr. Atakan Doğan for the useful comments he suggested for the thesis text.

I gratefully acknowledge the fundings we received from The Scientific and Technological Research Council of Turkey (TÜBİTAK) and Anadolu University Commission of Scientific Research Projects (BAP) under the grant numbers 111E053 and 1207F113, respectively.

I would love to present my deepest appreciation to my dear family who have never withhold their material and moral support during this thesis and my whole life.

I would like to dedicate this thesis to the precious memory of the beloved ones who were standing together with us on this perishable handful of soil and rock just at the beginning of this study...

Cihan Topal

August, 2014

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Eye tracking has emerged as an important research area with a diverse set of applications including human computer interaction, diagnosis of psychological, neurological and ophthalmological individuals, assistive systems for drivers or disabled people, marketing research, and biometrics. In medical research, for example, tracking and analyzing the saccades of a subject reveals important information about the psychological and neurological status of the subject, and enables the diagnosis of balance disorder, diabetic retinopathy, multiple sclerosis, etc.

Similarly, in marketing research, having answers to questions as "where a customer looks at first on a package of a product", or "which details on a poster tempt a person to watch the movie" provides valuable insight, and enables designing more attractive product packages or posters to increase commercial gains. In addition, eye tracking technology may speed up the educational procedures of the pilots, drivers or operators with useful findings attained by analyzing the gazes of the experienced ones. There are also many biometrics and security applications that benefit from the achievements of eye tracking research.

Estimation of the point-of-gaze (PoG) of a person in eye tracking research is an important problem that has been thoroughly worked for more than a couple of decades. Determination of where or how long an individual looks at provides valuable information for many scientific and practical aspects. In addition to the human computer interaction research, gaze information is also critical for the areas such as ophthalmology, psychology and neurology. Several real life applications that utilize the PoG estimation techniques can be listed in the following;

- assistive systems for disabled people (gaze controlled wheelchair, computer, etc.),
- diagnosis of various diseases by analyzing the gaze tracks of the user,
- education/assisting applications for pilots/drivers,
- virtual and mixed reality applications, etc.

Eye tracking systems can be classified with respect to several aspects, i.e., infrared-based or appearance-based, sensor-based or video-based, etc. Another prevalent classification aspect of eye tracking systems is their physical structure, i.e., wearable or remote, intrusive or non-intrusive. The choice of the eye tracker type highly depends on the application. Remote eye trackers may be more suitable for assistive systems designed for handicapped people due to the ease of deployment and use. On the other hand, using a wearable system may be more convenient if a scientific, medical or mobile task is at the hand. Since wearable systems can be deployed closer to the eye, they can obtain higher resolution, better quality eye images which results in more precise extraction of eye features. However, they require a further calibration to associate the point of gaze with the world coordinate system.

There are various methods for detecting the PoG in eye tracking systems; video-oculographic (VOG) methods, scleral search coil methods, electro-oculographic (EOG) methods and etc. Among these methods, VOG systems are usually preferred since they are less intrusive than the others and provide more comfortable use. In this study we aim to design and develop a VOG gaze estimation system which acquires and process input information as image streams of user's eye.

The most critical issue on the PoG estimation problem is designing the system in order to enable head movements for the user during tracking. The systems that do not support head-pose invariance and require users to keep

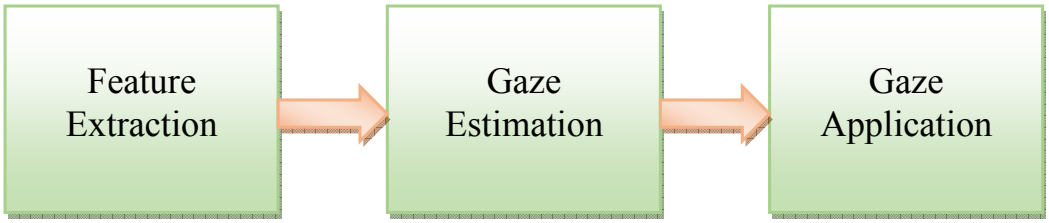


Figure 1.1: Three main components of a fully functional eye tracking system.

their head still while using the system and this situation dramatically reduces the comfort and efficiency of the experience.

A fully functional eye tracking system has three main components (see Fig. 1.1). Each of these components can be considered as different problems and studied individually in the literature. We explain each of these steps briefly in the following and give fine details in the consequent chapters.

1.1 Feature Extraction

In feature extraction step required information for gaze estimation method is gathered. This information can be voltage values from sensors in an EOG system, or pupil center detected from eye images in a VOG system. Widely used features on VOG systems are pupil center, cornea center and glints. The literature is very large on feature extraction methods for VOG systems. There are various illumination techniques and image processing algorithms for extracting the necessary information from eye images. In this thesis, we present a robust pupil boundary detection method that is able to detect the pupil even in tough occlusive situations. The algorithm starts by detecting and validating all edge segments in an image with Edge Drawing Parameter Free (EDPF) algorithm. Next, a modal structure estimation for each edge segment is performed by analyzing their gradient vector distributions. In this step, we try to find a near-circular segment that traverses the entire boundary of the pupil by maximizing the entropy of the gradient distributions. Once a near-circular segment is found, elliptical arcs are extracted from only

this segment. If no such segment exists, i.e., when the pupil is occluded by the eyelids or eyelashes; arcs from all segments in the image are extracted. Thereby, the algorithm adapts itself and reduces the computation time by trying to understand whether the pupil is entirely visible or occluded. Extracted arcs are then grouped to generate several pupil contour candidates, and one of them is chosen to be the final pupil contour by a cost function. The algorithm is also promising at detection of true negatives which is a useful property for many applications. The algorithm takes a mere 10 ms on non-occluded, and about 30 ms on occluded pupil images size of 640×480.

1.2 Gaze Estimation

Once the required information is obtained in feature extraction step, it is processed in the gaze estimation step to compute the location where the user looks at. Gaze estimation methods are classified into two main categories: 2D regression-based and 3D model-based, both have pros and cons. Regression-based methods easily approximate the non-linear characteristics of human eye, however, they are ineffective on providing head pose invariance. On the other hand, model-based methods are more successful on the same issue, yet they either require more hardware to accurately model the eye structure, or make assumptions on geometric parameters of eye. In this study, we propose a hybrid gaze estimation methodology which attempts to utilize the advantages of both regression-based and model-based gaze estimation methods. In other words, our method provides head pose invariance with minimal hardware and easy to implement since it does not require geometric calibration of the system. In our model, a head-mounted eye tracker is employed which both eases high quality eye feature extraction and accurate head pose estimation with the help of fiducial markers. Accordingly, a method to fuse geometric calibrations and regression parameters into gaze

coordinates is proposed. Another key contribution of this thesis is *hierarchical calibration* which is an easy procedure to enhance the gaze estimation accuracy. The experiments we performed show that the proposed method provides gaze estimation accuracy 0.37° in average, and easy to use in both assistive and mobile applications. Properties of 2D regression-based and 3D model-based approaches are criticized and an extensive comparison with the proposed method is provided in Chapter 3.

1.3 Gaze Application

The last part of a fully functional gaze estimation system is a gaze driven software which is particularly developed for a specific application. For example, computer applications are developed to be used with keyboard and mouse. In a similar vein, tablet and smartphone applications are developed to be used with finger taps and gestures. None of these devices cannot properly be used with gaze interaction. To provide efficient utilization of a computer or any other device, development of special applications is required. The most common type of these applications is on-screen keyboards which are developed to ease the text input by gaze. In these applications many assistive techniques such as word prediction and completion are applied to boost the text throughput.

1.4 Summary of Contributions

In this thesis we design and develop a complete gaze estimation system with all three components consisting of feature extraction, gaze estimation and gaze application as described above.

For the first part, we develop a real-time pupil boundary detection algorithm which is able to run even in tough occlusions of eyelids and eyelashes. The proposed algorithm also concerns real-time applicability by inferring

easy and tough cases with a heuristic. Therefore, it spends less time for the cases which the pupil is in clear sight.

In the second step, we design a novel eye gaze estimation model that incorporates the advantageous sides of conventional 2d regression-based and 3D model based methods. The proposed method both enables accurate gaze estimation with free head movement and does not require troublesome geometric calibration.

In the last part, we design and implement an on-screen keyboard that we name SliceType. SliceType is developed with a couple of design issues. First it aims to optimize the utilization of screen area by merging the regions of unused letters. In this way it reduces the erroneous letter selection ratio during gaze driven input. Second, it has integrated word completion and word prediction mechanisms to boost the word throughput.

We give the literature overviews and explain the contributions for each step of this thesis in the following chapters. More detailed information on eye tracking methods and related human-computer interaction topics can be found in the surveys [45, 46, 22].

2. PUPIL DETECTION

2.1 Introduction

All eye tracking methods start with the extraction of necessary features from face or eye. In this part of the thesis we aim to find an efficient method for detecting the pupil contour hence the center in a fast and efficient way. Once we detect the pupil and its center, we utilize that information in the consequent steps of the algorithm to estimate the PoG.

Pupil boundary detection and center estimation is an essential step in many applications. In applications such as eye tracking and gaze estimation, the detection of pupil center is required for the solution. On the other hand, pupil boundary detection is a harder problem and its accurate detection is indispensable for biometrics and medical studies. For any kind of application, the detection of boundary or center has to be performed precisely. For instance, even one pixel precision loss on pupil center may cause an error of a few degrees in the gaze direction vector, which would result in a significant drift in the estimated gaze point.

In this study, we propose a new robust and fast pupil boundary detection method from eye images, which works by extracting arcs from the edge segments of the image and joining them to find the pupil boundary, hence the center. The organization of the paper is as follows; we give a comprehensive related work on pupil boundary and center detection in section 2.2, we explain our method with fine detail in section 2.3, analyses of the method in terms of success and running time are presented with experimental results in section 3.6, and we finalize the paper with concluding remarks in section 3.8.

Table 2.1: A brief taxonomy for pupil boundary detection / center estimation algorithms.

REFERENCES	Downsampling	Bright / Dark Pupil Differencing	Image Thresholding / Binarization	Morphological Operations	Edge Detection	Blob Detection / Con. Comp. Ana.	Circle / Ellipse Fitting	Center of Mass Algorithm	Use of Temporal Information	Other / Comments
Ebisawa [1]		●	●	●				●		⌀
Morimoto et al. [51]		●	●						●	⌀
Hiley et al. [3]		●	●	iterative				●		⌀
Wang and Sun [4]			●	●	vertical		ellipse			♢
Gõni et al. [5]			adaptive	●				modified	●	⌀
Keil et al. [6]			●		for glint			●		⌀, ★
Agustin et al. [7]			●				ellipse			
Long et al. [9]			●			●		symmetric		⌀
Lin et al. [10]	●		●	●	●			parallelogram		⌀
Ma et al. [11]			adaptive		●		Hough			
Mäenpää [13]			●	●				●		♢
Dey and Samanta [14]	●				●	for edges	circle			★
Zhu et al. [15]			●			●	ellipse			△
Li et al. [16]					radial		ellipse			▼
Kumar et al. [18]				●	●	for edges				□

⌀ does not detect the pupil boundary, only estimates its center. ♢ performs iris detection.
★ applies histogram back-projection or non-linear power transform on the image to make the pupil more salient.
△ before ellipse fitting, tries to determine the false pupil contour pixels with respect to their curvature values by a set of assumptions.
▼ requires removal of glints. Assumes that the initial point for ray casting is inside the pupil. Iterative algorithm.
□ performs Fast Radial Symmetry detection and Delaunay Triangulation. Removes unwanted artifacts such as glints by a set of morphological assumptions.

2.2 Related Work

The literature on pupil detection is very rich, and many different techniques have been proposed. In this section, our goal is to give a high-level picture of the proposed solutions for pupil boundary detection and/or pupil center estimation.

One common technique for pupil detection is dark/bright pupil differencing [1]-[3]. In general, this technique is useful to roughly detect the eye locations in a remotely taken image. First, two eye images are captured with on-axis and off-axis illumination. Due to the physical structure of human pupil, on-axis illumination causes a significant brightness inside the pupil. Then difference image of two successive frames is computed and thresholded. Morphological operations are usually performed on the binarized image to visually enhance the result.

Wang and Sung [4] detect iris instead of the pupil in their gaze estimation study. They threshold the eye image and apply morphological opening to the pixel sets. Next, they find vertical edges, select two longest edge segments and fit ellipse. Goñi et al. [5] adaptively threshold the eye image and apply morphological open/close operations. Then connected component analysis is applied to detect the blobs. Later, blobs are filtered according to their sizes, and the most compact blob is selected as the pupil. Finally, center of the pupil is calculated with a modified center of mass algorithm. Keil et al. [6] first compute edge map and look for the glints to search the pupil nearby them. Dark pixels composing the pupil are extracted with a histogram back-projection, and the centroid of these pixels is considered as the pupil center. Agustin et al. [7] threshold the image and extract the points between pupil and iris. Then they fit ellipse to these points with RANSAC [8].

The occlusions caused by eyelids and eyelashes make detection of pupil

a lot more difficult and there are studies that specifically target this problem. Long et al. [9] threshold the eye images with a user-set value. Then, blob detection is applied and the largest blob is selected as the pupil. Pupil center is computed by placing a parallelogram into the pupil contour with the assumption that the centroids of the pupil ellipse and the parallelogram will overlap. In a similar vein, Lin et al. [10] use inscribed parallelograms to make a good estimate of the pupil center in occlusive cases. The image is thresholded and undesirable artifacts are removed by morphological operations. Next, pupil contour is extracted and three parallelograms are inscribed into it. If the centroids of these parallelograms are not overlapped, the pupil contour is rotated and the previous step is repeated.

Ma et al. [11] add up the intensity values of rows and columns, and take the lowest $[row, column]$ value as approximate pupil center in their iris recognition study. Then, adaptive thresholding is performed and a local image region is determined according to the approximated pupil center. Finally, edge detection followed by Hough circle transform [12] are applied.

Mäenpää [13] finds the pupil centre by thresholding and morphological opening in their iris study. Dey and Samanta [14] downsample image to reduce computation time, and apply a non-linear power transform to increase the contrast of the pupil. Then Gaussian smoothing followed by edge detection are performed. Next, edge segments are extracted and bounding box of each edge segment is found as a modal heuristic. Finally, circle fitting is applied to each segment to determine the pupil center.

There are also methods using alternative ways in feature extraction for pupil detection. In [15] authors propose using curvature of pupil contour to detect pupil boundary. After thresholding and blob detection are applied, small blobs are eliminated by a pre-defined threshold. Next, contour of

pupil's blob is extracted and its curvature is calculated. Finally, edge pixels of the pupil's contour are then classified by employing a set of rules (e.g. eyelids have positive curvature, etc.) and ellipse fit applied to the selected pixels.

Another study, Starburst, estimates pupil center by an iterative radial feature detection technique instead of finding all edges [16]. The algorithm starts by locating and removing the glint. Then, rays are cast from an initial point (assumed to be inside the pupil) with a 20° of radial step. Each ray stops when the derivation is greater than a threshold value, i.e., when a sharp intensity change occurs. This operation is iterated a few times with an updated starting point and a group of feature points are collected at each step. Finally, an ellipse is fit to the collected feature points with RANSAC [8]. Although the algorithm differs by its feature extraction scheme, it is not clear how it performs in occlusive cases as the pupil is in clear sight in all experiments presented in the paper. Authors also present a study that tries to adapt the Starburst algorithm to elliptical iris segmentation problem [17].

Another different approach on pupil detection is used in EyeSeeCam project [18]. The algorithm performs edge detection followed by connected component analysis to obtain edge segments. Thereafter, unwanted artifacts like glints and other reflections are deleted by applying a sequence of morphological operations and assumptions. Finally, Delaunay Triangulation is applied to remaining pixels and the pupil's boundary is detected assuming it is a convex hull. Although results of the algorithm are successful in occlusive cases, it has a complicated structure consisting of assumptions and morphological rules. Furthermore, there is no concrete information about the computation time requirements of the algorithm, except the authors' claim that all employed tools are computationally inexpensive.

In addition to above algorithms, there are also purely model-based methods which are mostly utilized in iris recognition studies. As an example, Daugman [19] proposes an integro-differential operator for detecting the pupil and iris boundaries aiming to maximize the contour integral value on the smoothed image derivative (Eq. 2.1).

$$\max_{(r,x_0,y_0)} \left| G_\sigma(r) * \frac{\partial}{\partial r} \oint_{(r,x_0,y_0)} \frac{I(x,y)}{2\pi r} ds \right| \tag{2.1}$$

Arvacheh and Tizhoosh [20] developed an iterative algorithm based on an active counter model which is also capable of detecting near-circular shapes such as the iris and the pupil. These methods work fine; however, they require a full search of the image plane in order to find r, x_0, y_0 parameters that maximize the given model. This search operation is computationally expensive and highly vulnerable in cases where the pupil is not perfectly circular. Therefore, they cannot be employed in real-time eye tracking applications.

Table 2.1 gives a brief taxonomy of the above-mentioned pupil detection algorithms. Before we go over the table and describe relative merits or disadvantages of the employed methods, it is important to remind that even small errors in pupil center or boundary detection can cause significant inefficiencies in medical or gaze applications. Therefore, downsampling the image to save computational time is obviously undesirable as it decreases the accuracy by wasting fiducial information. Next technique on the table is the bright/-dark pupil differencing. Although this technique takes little computation and eases roughly locating the pupil, it has important disadvantages. Firstly, it needs additional hardware to obtain bright and dark pupil images consecutively in a synchronous manner. Secondly, it reduces the temporal resolution in half since it needs two frames to perform single detection. Thirdly, it is very sensitive to motion; that is, if a large pupil displacement occurs between

two consecutive images, the algorithm fails. Ebisawa specifically addresses this problem in [21], and proposes various methods for positional compensation.

As seen from Table 2.1, thresholding is the most common technique in pupil detection. Although thresholding can quickly discriminate image regions with different intensity values, it is highly vulnerable to lighting conditions and parameter selection. Consequently, it fails on finding the exact location where intensity change occurs and decreases the accuracy. Another commonly employed technique is morphological operations which are applied on the thresholded binary image to improve visual structure of the image. However, they may also affect the actual information on the image and cause significant errors on the result. Moreover, the algorithms which utilize bright/dark pupil, thresholding and blob detection to find a center point for pupil are obviously not capable of detecting the boundary. Hence, they cannot be employed in most biometrics or medical studies which require precise detection of the boundaries of pupil and iris.

In this section we presented an overview of related studies covering both biometrics and eye tracking areas from the viewpoint of pupil detection problem. For interested readers, there are also comprehensive surveys that review the gaze estimation literature, in particular [22, 23].

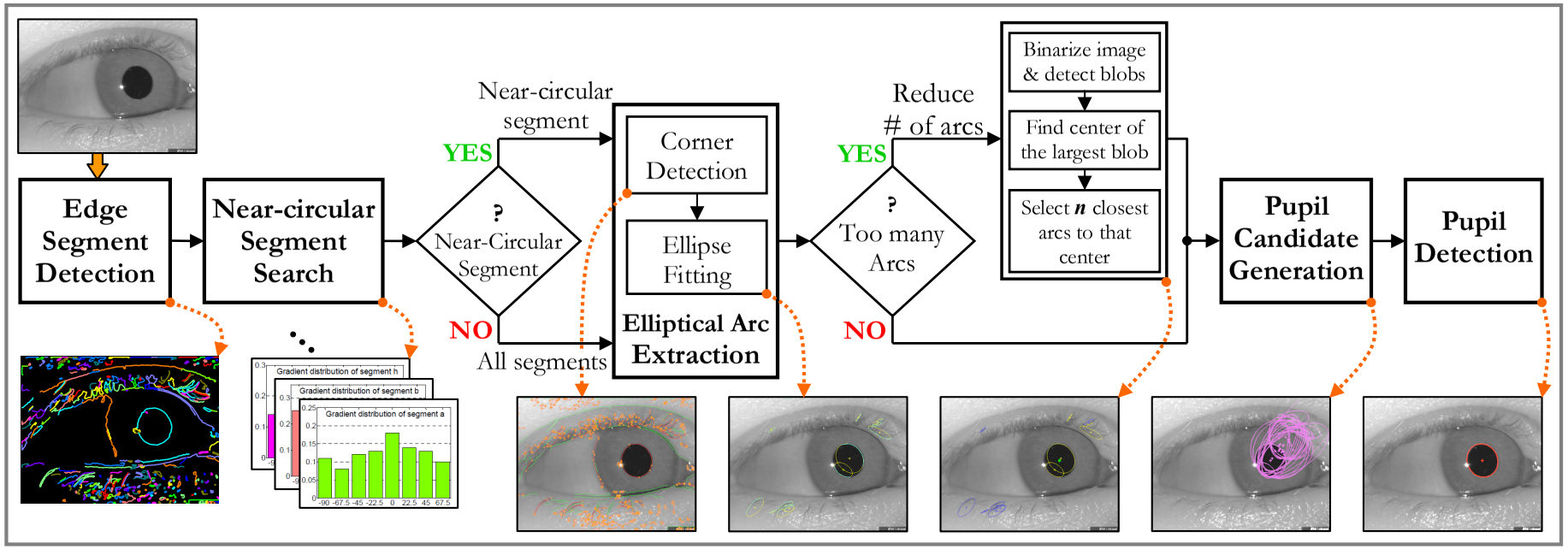


Figure 2.1: Processing pipeline of the proposed algorithm.

2.3 Proposed Method

In this study, we propose an adaptive method for pupil boundary detection which is able to save computation by inferring whether an occlusion is the case or not. When the pupil is in clear sight, the computation takes very little time; when the pupil is severely occluded, the algorithm infers it and spends more effort to detect the pupil without compromising real-time applicability. The main strategy improving the method against the occlusive cases is extracting the elliptical arcs from the image and finding one arc or a group of arcs representing the pupil contour. In this way, relevant features from a partially visible pupil can be extracted and detection can be performed by fusion of separate features. Besides detecting the pupil boundary and center precisely, the algorithm can also identify if there is no pupil in the image as a very handy information for many applications.

The proposed method has a simple flow and consists of the processing pipeline shown in Fig. 2.1. The processing starts by the detection of the edge segments by the Edge Drawing (ED) algorithm. ED returns a set of edge segments, each of which is a contiguous chain of pixels.

After the edge segments are extracted, the next step is to determine whether a near-circular segment exists that traces the entire boundary of the pupil. Such an edge segment would be found if the pupil is clearly visible with no or very little occlusion. To find whether an edge segment has a circular geometry, we devise a fast heuristic method which is based on the gradient distribution of the segments. On the condition that a near-circular segment is found, we extract elliptical arcs only from that segment. If no near-circular segment is found, which would be the case if the pupil is severely occluded by the eyelids or eyelashes, then arcs from all edge segments in the entire image are extracted.

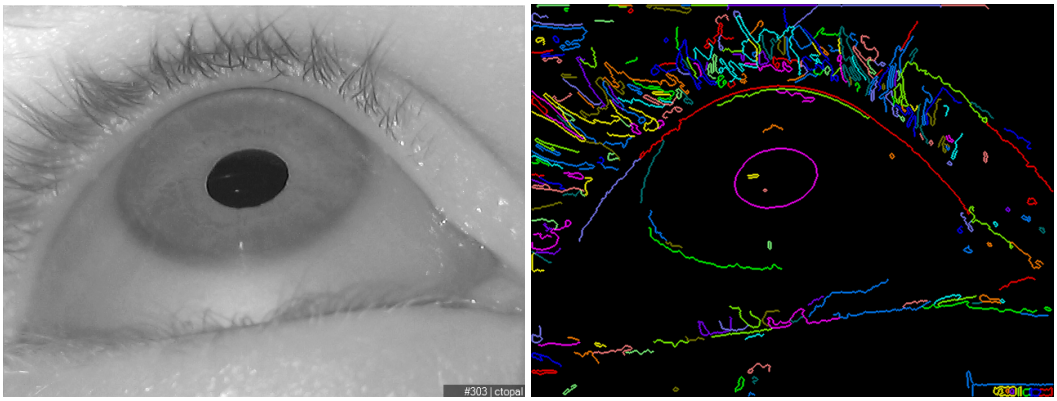
Following the extraction of the elliptical arcs, the next step is to join the arcs to generate a set of ellipse candidates that trace the pupil boundary. Candidates are finally evaluated for their relevance to be the actual pupil contour and the best one, if it exists, from among the candidate ellipses is chosen.

In the following subsections, our goal is to elaborate each step of the proposed algorithm in detail, giving enough detail to make the discussion clear.

2.3.1 Edge Segment Detection

To detect all edge segments of an image, we employ our recently proposed edge/edge segment detector, Edge Drawing (ED) [24]-[27]. Unlike traditional edge detectors which work by identifying a set of potential edge pixels in an image and eliminating non-edge pixels through operations such as morphological operations, non-maximal suppression, hysteresis thresholding etc., ED follows a proactive approach. The ED algorithm works by first identifying a set of points in the image, called the anchors, and then joins these anchors using a smart routing procedure; that is, ED literally draws edges in an image. ED outputs not only a binary edge map similar to those output by traditional edge detectors, but it also outputs the result as a set of edge segments each of which is a contiguous and connected pixel chain [26]. This property of ED extremely eases the application of the algorithm to further detection and recognition problems.

Similar to all edge detectors, ED has many parameters that must be tuned by the user for different images. Ideally, one would want to have an edge detector which runs with a fixed set of parameters for any type of image. To achieve this goal, we have recently incorporated ED with the a contrario edge validation mechanism due to the Helmholtz principle [28, 29],



(a) (b)

Figure 2.2: **(a)** Sample eye image, **(b)** 255 edge segments obtained by the EDPF algorithm. Each color represents a different edge segment.

and obtained a fast parameter-free edge segment detector, which we name edge drawing parameter free (EDPF) [30, 31]. EDPF works by running ED with all ED's parameters at their extremes, which detects all possible edge segments in a given image with many false positives. We then validate the extracted edge segments by the Helmholtz principle, which eliminates false detections leaving only perceptually meaningful edge segments with respect to the a contrario approach with very little overhead of computation. Fig. 2.2 illustrates detected edge segments for an example eye image. In the figure, each color represents a different edge segment, which is one-pixel wide, contiguous chain of pixels.

2.3.2 Near-Circular Segment Search

The main goal of this step is detecting the pupil in an easy and computation efficient way when it is entirely visible without any occlusions. Once we have the edge segments detected, we need to find the one that traverses the pupil boundary. The first thing that comes to mind is to apply a brute force search as follows: fit an ellipse to each edge segment, compute the fitting error, and pick the edge segment that has the smallest fitting error. This method might work only when the pupil is clearly visible, i.e., when the pupil is not occluded

by IR-LED glints or eyelashes, however, fitting an ellipse and calculating the fitting error for each segment requires too much computation.

To reduce this computational burden, we devise a faster method based on the analysis of gradient directions to find the near-circular segment, if one exists. Gradient information of the segments contains substantial information about the geometrical structure and is used in shape matching, retrieval and recognition problems [32]-[36]. Since we already have the vertical and horizontal derivatives of the image computed during the edge detection scheme, we can find the gradient directions with simple trigonometry in a computationally cheap way (see Fig. 2.3.a). The *arctan* function obviously results angle values in an interval $[-\pi/2, \pi/2]$, providing an angle range of 180°. Before examining the distribution of gradients, we quantize the angles with 22.5° to obtain discrete symbols, thus we divide the unit circle into 16 regions into 8 different directions (see Fig. 2.3.b).

Once we get the quantized gradient directions for all pixels in each segment, we try to infer the modal characteristics of each segment by analyzing their gradient distributions. It is easy to observe that any segment in near-circular form would have an even gradient distribution if the tangential gradients on its perimeter would be sampled with a fixed angular step.

Fig. 2.4 shows the edge segments of an eye image and the distributions of the gradient directions for 9 edge segments labelled a-i. As seen from the figure, circular edge segments have an even distribution; whereas, straight edge segments have an unbalanced distribution, where a few values dominate.

Thus, to distinguish the segments having circular shapes, we need to select the segments that result in plain gradient distribution. To achieve this, we use the entropy function (Eq. 2.2) on the quantized gradient distributions

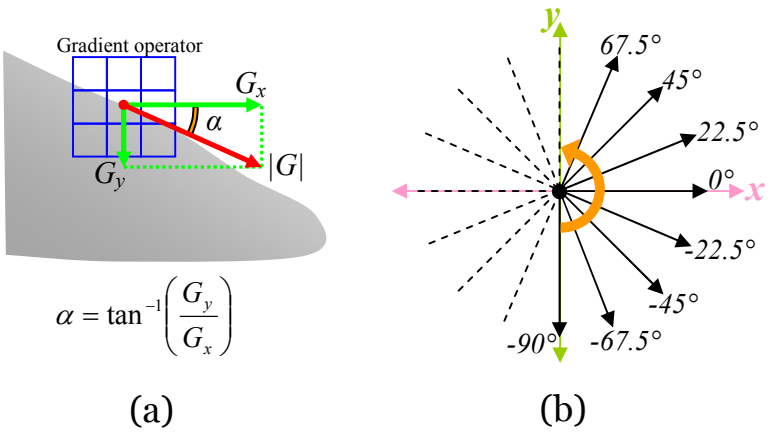


Figure 2.3: (a) Gradient computation with image derivatives, (b) Quantization of computed gradient directions.

of the segments.

$$E = -\sum_{i}^{n} p_i . \log(p_i) \tag{2.2}$$

Since entropy function maximizes for flat distributions where the frequency of each symbol is equal, we compute the entropies of the gradient distributions of the segments (Eq. 2.3).

$$\arg\max \left| \sum_{i}^{8} f_{G_i} . \log(f_{G_i}) \right| \tag{2.3}$$

where f_{G_i} is the frequency of the i^{th} gradient direction. The entropy values for the segments are maximized for a perfect circle and gets lower as the segment shape differs from being a circle (elliptic, etc.), and finally entropy becomes zero for straight lines since a straight line has only one gradient direction along its geometry. Since the number of different symbols is 8; the maximum entropy value is $\log_2 8 = 3$ in our case. In Table 2.2, the lengths and the entropy values of the segments shown in Fig. 2.4 are listed. It is easy to observe that circular edge segments have higher gradient entropy values regardless their size, whereas straight edge segments have lower values as expected.

With this heuristic, we can discard the segments producing small en-

tropy values than a certain threshold in a fast manner. When examining speed of the heuristic, we measure that computing the gradient entropy of an edge segment with available image derivatives is 150~200 times faster than ellipse fitting and error computation according to length of the segment. In this way, we avoid spending computation time on the segments which have irrelevant geometries.

Following the computation of the edge segment entropies, one segment is chosen to be the near-circular segment to extract elliptical arcs if it satisfies the following three conditions:

i) Must have a very high gradient entropy. The theoretical entropy upper-bound for 8 different gradient directions is $\log_2 8 = 3$. Accordingly, we choose the segments which have 2.9 or more gradient entropy.

ii) Must have a small ellipse fitting error, e.g., 2 pixels, when an ellipse is fit to the pixels of the segment.

iii) Must be a closed segment. To avoid problems due to the small occlusions, such as glints, we consider a 15 pixels threshold for the distance between the start and end points of the segment.

Evaluation of the second condition requires ellipse fitting to a set of points, for which we employ the method proposed by Fitzgibbon et al. [37]. To compute the ellipse fitting error, there is no straightforward method; so, we developed a method based on [38]. We estimate the distance between the ellipse and each point by solving the equations described in [38] with Newton-Raphson method in a couple of iterations. Once we estimate all distance values between each point and the ellipse, we calculate the normalized rms distance to obtain a single scalar to represent the fitting error. For ellipse perimeter calculation, which also does not have an exact solution, we employ Ramanujan's

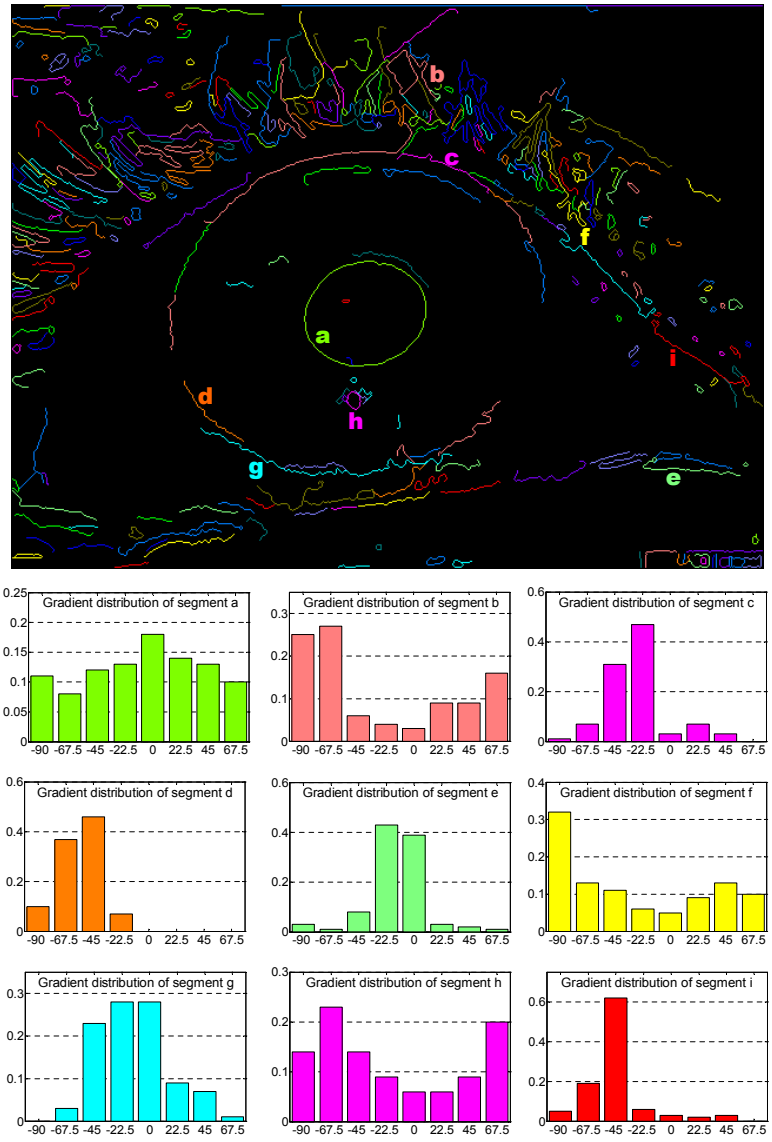


Figure 2.4: Detected edge segments displayed with different colors (top), Gradient distributions of the labeled segments in the image (a to i).

second approximation [39]. In the event that more than one edge segment satisfies all three conditions given above, the one having the minimum ellipse fitting error is chosen to be the near-circular segment. While existence of a near-circular segment speeds up the computation, its not compulsory for the detection of pupil.

It is important to note that shape of a segment does not necessarily have to be near-circular to give high entropy values. In addition to segments with near-circular geometry, segments which have concave shapes or follow

complex trajectories can also produce high entropy values. Therefore, we use entropy test as a prerequisite to accelerate the algorithm and make final decision about a segment by ellipse fitting.

2.3.3 Elliptical Arc Extraction

The next step of the algorithm is extracting the elliptical arcs (which will be referred to as *arc* hereafter) from the gathered edge segments. If a near-circular segment could be found at the previous step, arcs are extracted only from that segment. If no near-circular segment is found, then all segments are subjected to arc extraction process. In this manner, the algorithm adapts itself and requires less computation when the pupil contour is entirely visible.

In a previous work, we extract circular arcs by combining consecutive line segments to detect circles in an image [40, 41]. However, pupil's projection onto the camera plane can be in a more elliptical structure, hence we need to detect the elliptical arcs in this study. To solve this problem, we devise another strategy which also works fast. Finding the start and end points of a potential elliptical arc inside an edge segment is achieved by locating the corners along the segment. First, we detect corners on the segments with a fast curvature scale-space (CSS) method [42]. To gain speed, we used the same gradient information employed in previous steps to compute the turning angle curvature and detect corners. Afterwards, we fit an ellipse to the points lying in between two consecutive corners along each segment and obtain elliptical arcs.

Fig. 2.5.c and Fig. 2.5.d presents the results of our arc extraction method for 4 test images. In the first two rows, the pupil is completely visible; hence, the near-circular segment (the orange colored segment) is detected. Therefore, arc extraction is applied only to this segment. When no near-circular segment is found due to occlusions, arcs are extracted from all

Table 2.2: Lengths and gradient entropies of the segments shown in Fig. 2.4.

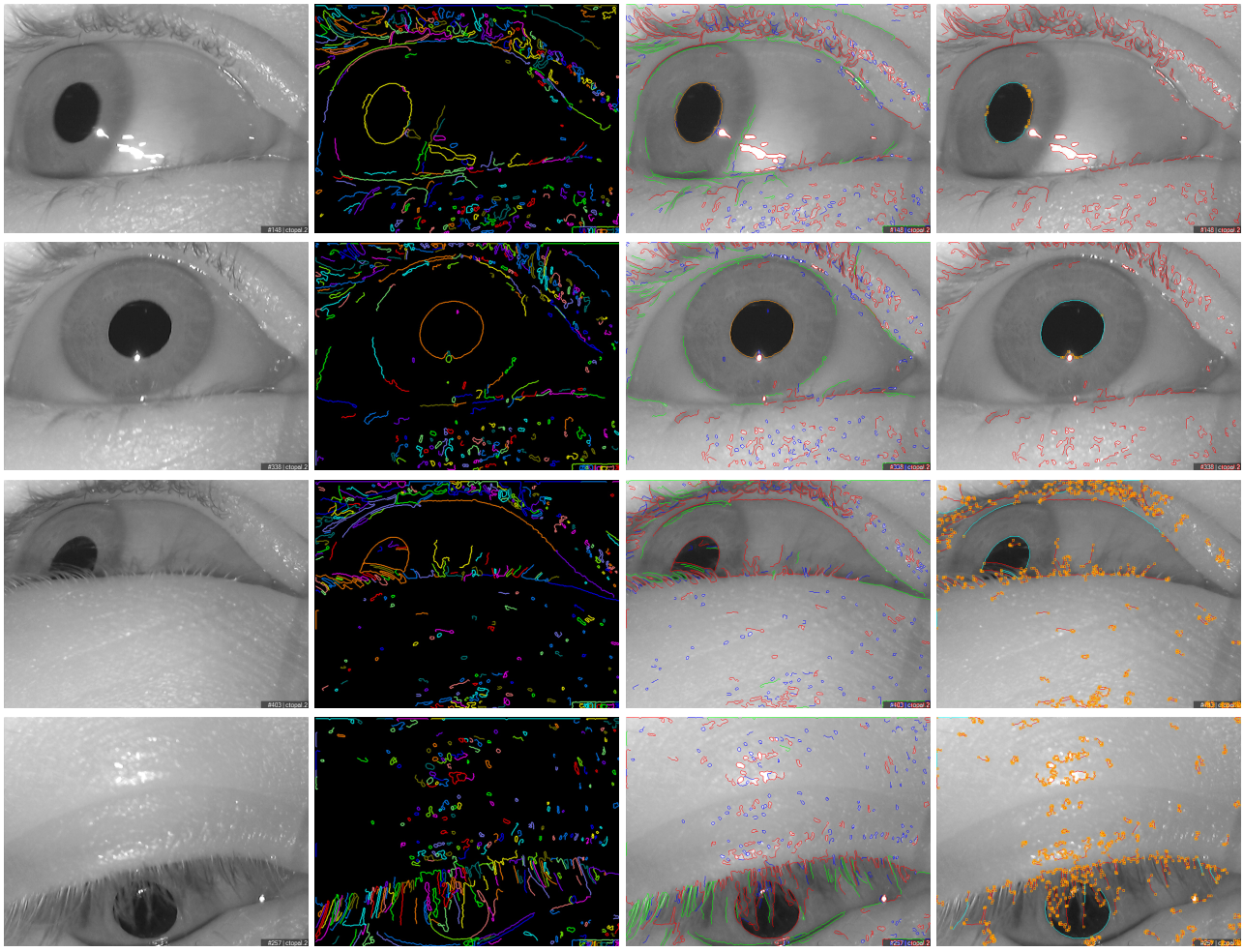
Segment ID.	Length (pix.)	Entropy
a	271	2.96
b	128	2.66
c	86	1.99
d	68	1.66
e	100	1.89
f	127	2.76
g	232	2.36
h	35	2.83
i	113	1.74

segments to avoid missing any useful information (see 3^{rd} and 4^{th} rows of Fig. 2.5). To save further computation time, we omit the segments having low gradient entropy (i.e., < 2) because their straight geometry rarely contains elliptical arcs. We also neglect the short segments (i.e., < 25 pixels) since they do not provide any useful information. In the third column of Fig. 2.5, low gradient segments and short segments are indicated in green and blue, respectively, and the final segments that are subjected to arc extraction displayed in red. The last column of Fig. 2.5 shows the detected corners (orange colored squares) and extracted arcs (colored cyan) for all images.

It should be noted that even if the pupil is in clear sight, it may appear highly elliptical because of the view angle. In such a case, pupil's segment may result in a low gradient entropy and near-circular segment would not be detected. As a consequence, elliptical arc extraction would be applied to all segments even though there is no occlusion.

2.3.4 Generation of the Pupil Candidates

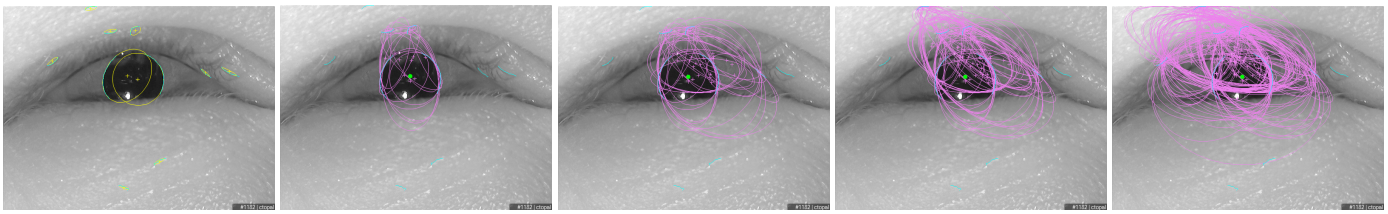
In this step, we generate candidate ellipses by grouping the extracted arcs. Thus we aim to detect the complete pupil boundary even if its boundary is partially visible. To generate pupil candidates, we try to fit an ellipse to each subset of all extracted arcs. Excluding the empty set, there are $2^n - 1$



(a) (b) (c) (d)

Figure 2.5: Arc extraction process. First two rows include examples where the pupil is clearly visible; whereas, in the last two rows the pupil is severely occluded. (a) input images, (b) detected edge segments displayed in different colors, (c) short segments, low gradient entropy segments and segments from which arcs will be extracted are colored in blue, green and red, respectively. The near-circular segment displayed in orange, if it exists, (d) corners (orange boxes) and extracted arcs (cyan segments). Note that if the near-circular segment is found, corner and arc detection is only applied to that segment.

different arc combinations for n arcs. Due to the exponential grow rate, the number of generated pupil candidates increases excessively for the number of extracted arcs and testing all combinations becomes infeasible. In Fig. 2.6 an example is shown to illustrate the relation between number of selected arcs and generated pupil candidates. As seen from the example, the number of generated pupil candidates increases in an excrescent manner for higher



(a) (b) (c) (d) (e)

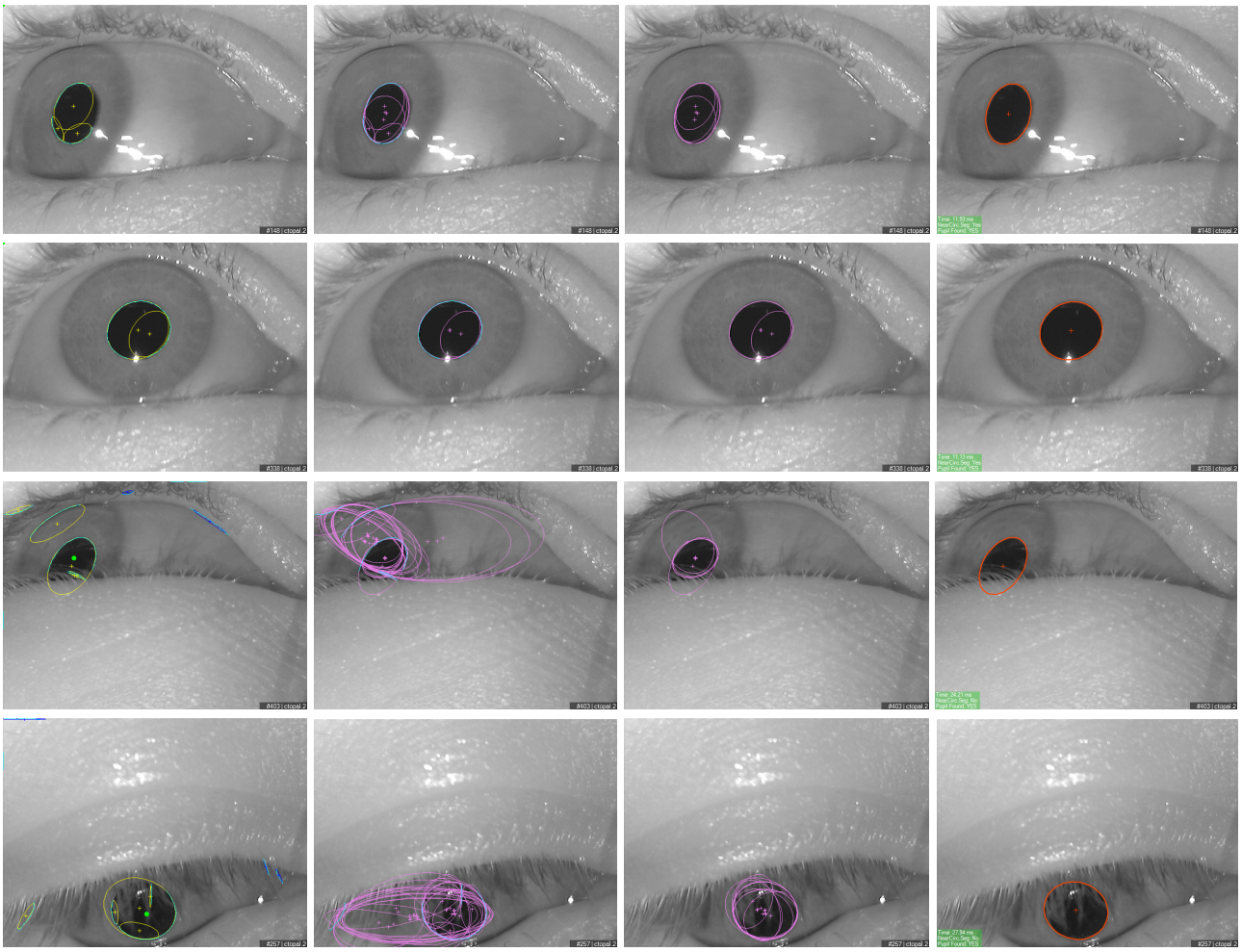
Figure 2.6: Generated pupil candidates for different number of arc selections. (a) Eye image with 10 elliptical arcs detected. (b)-(e) Generated 15, 31, 63, 127 pupil candidates for 4, 5, 6, 7 selected arcs, respectively. Remember that we generate $2^n - 1$ ellipses for each pupil candidate for n selected arcs.

number of selected arcs.

For the cases where the number of extracted arcs exceeds, we apply a fast selection method to reduce the number of arcs. Similar to many pupil center estimation methods, we simply binarize the image and find the centroid of the largest blob as a very rough estimation of pupil center. Then, we select a predefined number of closest arcs to that center.

In the experiments, we see that the pupil counter is detected as a couple of separated arcs even in tough occlusions, hence we select 5 arcs to guarantee reconstruction of the pupil boundary. If the number of extracted arcs is below 5, we do not perform arc selection which is the common case if there is no occlusion. In the last two rows of Fig. 2.7.a, selected and eliminated arcs are indicated with yellow and blue ellipses, respectively. The arcs displayed with yellow ellipses are selected with respect to the distance between their center and rough pupil center estimation (indicated with green circle). Please note that any arc elimination is not applied to the first two rows due to the small number of (i.e., ≤ 5) extracted arcs.

Once we obtain arcs, we analyze all $2^n - 1$ arc subsets by ellipse fitting to all arc pixels in each subset [37]. Fig. 2.7.b shows all generated pupil candidates for the input images in Fig. 2.5.a. Since the pupil candidate generation process considers all subsets of selected arcs, groups of unrelated



(a) (b) (c) (d)

Figure 2.7: Pupil candidate generation and pupil detection steps. First two rows include examples where the pupil has clear sight; whereas, last two rows present occlusive cases. (a) Detected pupil blob center is indicated with green spot. Ellipses show fitting results of arcs and displayed in yellow and blue for selected and eliminated ones, respectively. If the number of arcs exceeds a specific value (5 in the example), arcs selected with respect to the distance between the ellipse center and the pupil blob center. (b) $2^n - 1$ pupil candidates are generated by joining all possible arcs combinations, (c) remaining pupil candidates after elimination due to high fitting error, (d) the selected ellipse representing the pupil contour using the argument in Eq. 2.4.

arcs which do not form a valid conic geometry are also subjected to ellipse fit operation. Therefore, the next step is to eliminate those candidates having a high fitting error, which clearly cannot be the pupil boundary segment. Fig. 2.7.c shows the remaining ellipse candidates after this elimination. The

real pupil boundary segment will be chosen from among these remaining candidates.

2.3.5 Detection of the Pupil

In the previous step, we get a number of pupil candidates each of which consists of one subset of elliptical arcs. Accordingly, we still need to select one candidate ellipse to be the final pupil contour. To make the decision, we define a cost function which considers the following properties of a candidate ellipse:

i. the ellipse fitting error (ε),

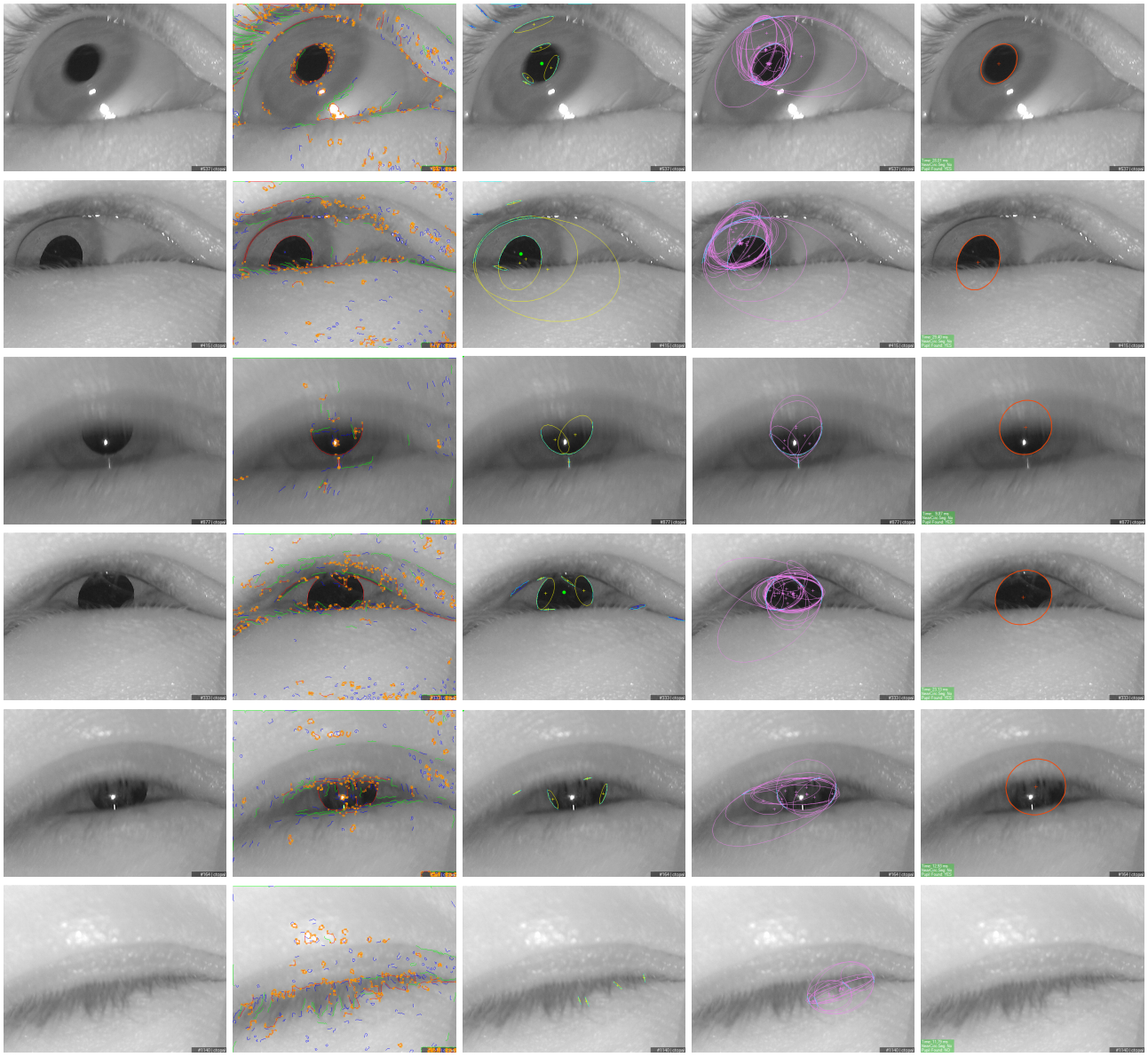
ii. the eccentricity (τ),

iii. the ratio of the arc pixels to the perimeter of resulting ellipse (ϕ).

Each of the pupil candidates is formed by one or more arcs. If the pupil boundary is detected with multiple arcs, the fitting error should be reasonable because we expect the arcs to be parts of the same contour. Thus we need to minimize the fitting error ε.

The eccentricity (τ) indicates the compactness of an ellipse and is the ratio of the semi-major and semi-minor axes. For a circle, it is obviously 1. We obtain diverse pupil candidates whose eccentricities can vary, i.e. $0.1 \sim 10$. However, pupil's projection onto the image plane is usually closer to a circle rather than a skewed ellipse in related applications. Therefore, we tend to select a candidate having an eccentricity close to 1.

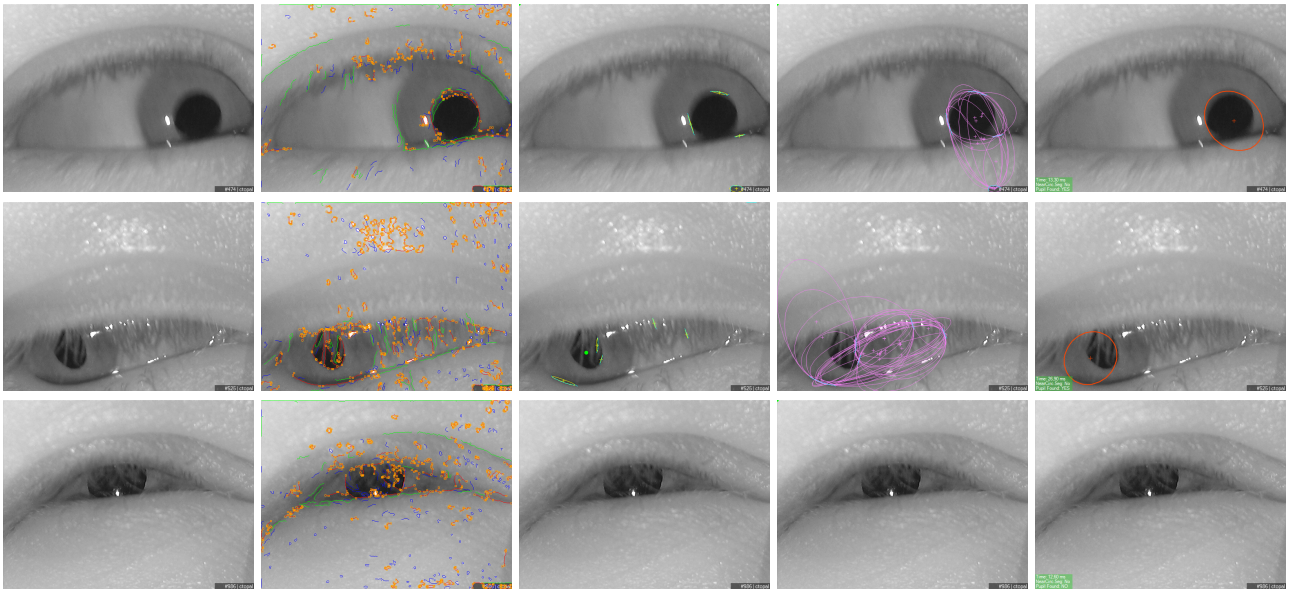
The parameter ϕ is the ratio of the number of pixels involved in ellipse fitting to the perimeter of the resulting ellipse. In some circumstances, one single and short arc may result in a large pupil candidate ellipse that may lead to inconsistency. Therefore, we look for the pupil candidates which are formed by consensus of more arc pixels and have larger ϕ.



(a) (b) (c) (d) (e)

Figure 2.8: A set of experimental results where the algorithm succeeds. (a) input image, (b) edge segments and corners, (c) extracted arcs pixels (cyan), selected arcs (yellow), eliminated arcs (blue) and pupil blob center (green), (d) generated pupil candidates, (e) the selected pupil candidate. Frame IDs and other related information are printed at the bottom of the images.

During our experiments, we observed that the effect of the eccentricity (τ) is less than the effect of ε and ϕ due to the possibility of true pupil not being the most compact ellipse among the candidates. Accordingly, we take the squares of ε and ϕ to increase their effect on the cost function. Finally,



(a) (b) (c) (d) (e)

Figure 2.9: A set of experimental results where the algorithm fails. Please refer to section 2.4.1 for the explanation of the causes of failures.

we need to minimize ε and τ and maximize ϕ in our formulation, and select the candidate that minimizes the following cost function:

$$\arg\min_{(\varepsilon,\tau,\phi)} \left| \frac{\varepsilon^2 \times e^{|1-\tau|}}{\phi^2} \right| \tag{2.4}$$

Fig. 2.7.d. shows the pupil detection results. Among the pupil candidates shown in Fig. 2.7.c, the one that minimizes cost in Eq. 2.4 is selected as the pupil.

2.3.6 Detection of True Negatives

In many applications, having the information that there is no pupil in the image is important as much as detecting the pupil. This information can provide very useful extensions to the eye tracking applications such as blink detection. It is possible to obtain arcs and pupil candidates although there is no pupil with the algorithm (see last row of Fig. 2.8). We observe that the cost function overshoots in these circumstances due to large ε and small ϕ values. Therefore, we can easily find the lack of the pupil by examining the

functions output. In Fig. 2.10 we present a plot of the cost function versus a number of frames sampled from an eye blink operation. It is seen that the function's output increases as the visible part of the pupil is getting smaller.

2.4 Experimental Results

In this section, we present results for a comprehensive set of experiments. First, we give pupil detection examples with challenging occlusive cases followed by cases where the algorithm fails. Then we analyse the running time of the algorithm by giving details for each step in both easy and tough cases. Finally, we test the accuracy of the algorithm with a simple gaze estimation application. Besides the results given in this section, we present videos and more results on our website [43].

2.4.1 Detection Performance

In the experiments we used a mobile eye tracker built with two off-the-shelf webcams shown in the video at `<http://goo.gl/wh1NBL>`. Fig. 2.8 shows the results for a number of cases where the algorithm succeeds in detection of the pupil even when the pupil suffers from motion blur (1^{st} row), or it is occluded from above, below and from both sides. In the last row the eye is closed, and the algorithm correctly concludes that there is no pupil in the image.

Fig. 2.9 shows the results for cases where the algorithm fails. The failures can occur as false detections or misses. The common reasons for the failures can be listed as follows:

- failure on detecting edge segments from pupil contour due to the motion blur or high level occlusion,
- failure on arc detection due to inaccurate corner detection,
- failure of the cost function in selecting the correct pupil candidate.

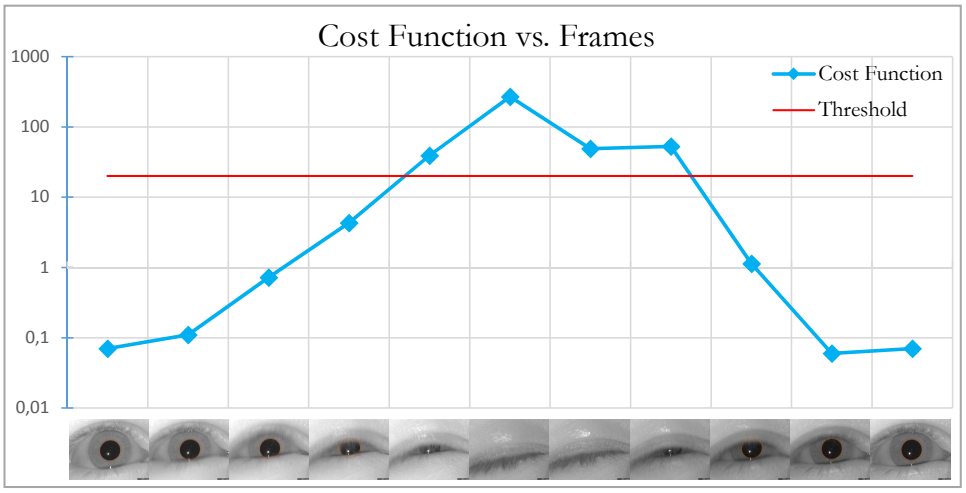


Figure 2.10: Output of our cost function for a set of example frames. It is seen that the result of cost function increases proportional to the pupil occlusion. If the occlusion is dramatic or there is no pupil in the image, the value of the function overshoots. (Notice that the plot is in logarithmic scale.)

We also present a video consists of 1600+ frames including blinks and occlusive cases at <http://goo.gl/z2VugA>.

2.4.2 Running Time Analysis

Table 2.3 gives a dissection of the running time of the algorithm for the images given in Fig. 2.8 and Fig. 2.9. The running time of the algorithm strictly depends on the detection of the near-circular segment in the first place. If the near-circular segment is found, the algorithm adapts itself and runs much faster. The most time-consuming routines of the algorithm are ellipse fitting and error computation methods and they are being invoked intensively during the execution. Especially, fitting error computation method [38] estimates the distance between each point and the ellipse by an iterative approach and its computational cost strictly depends on the total length of extracted arcs in terms of pixels. On the other hand, edge segment detection and gradient entropy computation steps take the same amount of time regardless of the input image as seen in Table 2.3.

In Fig. 2.11.a and Fig. 2.11.b we illustrate the distribution of aver-

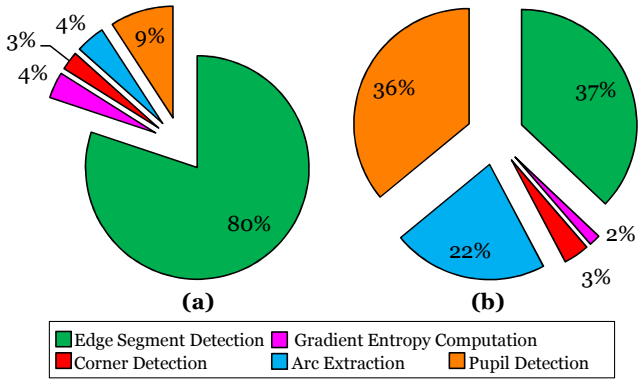


Figure 2.11: Distribution of execution times among the steps of the algorithm: (a) Easy (non-occlusive) cases, (b) Tough (occlusive) cases.

age execution times for clear and occlusive cases, respectively. As seen in Fig. 2.11.a, when the pupil is in clear sight, most of the time is spent on edge segment detection and the rest takes very little time. When the pupil is severely occluded and the near-circular segment cannot be found, then arc extraction and joining take more time than edge segment detection (see Fig. 2.11.b). We can say that the average running times for the images in Fig. 2.11.a and Fig. 2.11.b are roughly 10 ms and 30 ms on an 2.8 GHz Intel CPU.

We also prepared a demo video to clearly show how the algorithm adapts itself and runs faster when a near-circular segment is found. In the

Table 2.3: Running times (in ms) for examples in Fig.s. on an Intel 2.8 GHz CPU

Input Image	Edge Segment Detection	Gradient Entropy Comp.	Corner Detection	Arc Extraction	Pupil Detection	**TOTAL** (ms)
Fig.2.7 - 1^{st} row	9.65	0.92	0.07	0.34	0.95	**11.93**
Fig.2.7 - 2^{nd} row	9.22	0.85	0.05	0.35	0.66	**11.13**
Fig.2.7 - 3^{rd} row	9.17	0.43	1.21	5.66	7.74	**24.21**
Fig.2.7 - 4^{th} row	11.48	0.57	1.38	5.44	9.08	**27.94**
Fig.2.8 - 1^{st} row	10.69	0.48	0.97	7.74	8.12	**28.01**
Fig.2.8 - 2^{nd} row	9.50	0.29	0.73	5.40	13.49	**29.40**
Fig.2.8 - 3^{rd} row	7.98	0.12	0.19	0.43	1.15	**9.87**
Fig.2.8 - 4^{th} row	11.39	0.37	0.78	3.24	7.35	**23.13**
Fig.2.8 - 5^{th} row	9.64	0.26	0.55	0.76	1.72	**12.93**
Fig.2.8 - 6^{th} row	9.05	0.29	0.65	0.61	1.20	**11.79**

demonstration, we used a synthetic object and indicated the gradient entropy inside the detected boundary. When near-circular segment cannot be found due to the view angle or occlusion, arc extraction is applied on many segments and running time increases dramatically. More detail is provided on the video explanation at <http://goo.gl/tky8Zr>.

2.4.3 Accuracy

After visually inspecting the algorithm and analyzing the timing results, we incorporated the algorithm into a point-of-gaze (PoG) estimation application to obtain quantitative accuracy results. We performed a calibration procedure for 9 screen points and employed a second-order non-linear mapping with the following equations:

$$x_g = a_{x_4}x_p^2 + a_{x_3}y_p^2 + a_{x_2}x_p + a_{x_1}y_p + a_{x_0} \tag{2.5}$$

$$y_g = a_{y_4}x_p^2 + a_{y_3}y_p^2 + a_{y_2}x_p + a_{y_1}y_p + a_{y_0} \tag{2.6}$$

where (x_p, y_p) is detected pupil center, (x_g, y_g) estimated screen coordinate of gaze and $\boldsymbol{a_x}$ and $\boldsymbol{a_y}$ vectors are the calibration parameters for x and y coordinates, respectively.

In the experiments, we obtained an average gaze error about 0.3°. Since our system does not compensate the head movements yet, we used a chin-rest during the PoG estimation experiments. Although keeping the head still on a chin rest eases the problem, obtained accuracy value is still a neat achievement compare to the existing methods [22]. In Fig. 2.12 PoG error results for 10 different calibrations is presented. Additionally, we present a video including both calibration and tracking schemes of our gaze estimation experiment at <http://goo.gl/RVt2sz>. In another video, the user is entering text with our virtual keyboard design [44] at <http://goo.gl/QMOmuo>.

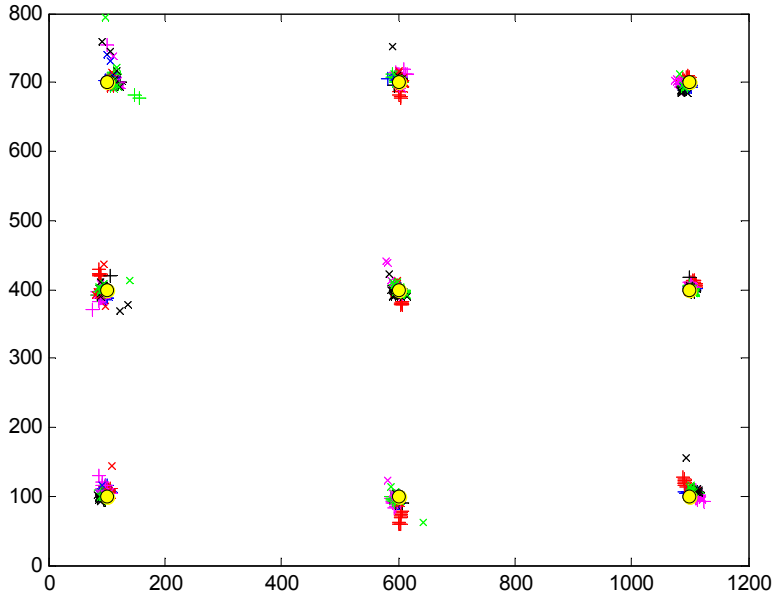


Figure 2.12: Point-of-gaze estimation errors for 10 different calibrations. Screen calibration points are displayed as yellow circles and each calibration result is indicated with different marking and color.

2.5 Conclusions

Pupil detection is an indispensable step in many eye tracking applications and has to be performed precisely. In most studies, pupil detection is handled with straightforward methods which lack accuracy and fail in occlusive cases. In this study we focused on developing a state of the art algorithm for pupil boundary detection and center estimation. We basically find the arc segments in the image and try to obtain a final ellipse encircling the pupil with the consensus of all obtained features.

Because the edge segment detection method we used has optimum localization, extracted arcs represent the pupil boundary, hence its center, very precisely. This ability of the algorithm can be useful especially in medical applications. Another important property of the algorithm is the proposed heuristic based on the distribution of gradients. In this way, the algorithm can adapt itself and run a couple of times faster if there is no or very less occlusion.

As the experimental results show, the proposed algorithm can detect the pupil even in tough occlusive cases without compromising the real-time applicability constraints. According to results, the algorithm is able to detect the pupil boundary even when 25% of its contour is visible by the camera.

3. GAZE ESTIMATION

3.1 Introduction

In this part of the thesis study we propose a method for estimating the point-of-gaze with the obtained image features in the previous step. Our purpose in this step is to design a gaze estimation model which merges the upsides of both 2D regression-based and 3D model-based approaches. In this way we investigate the possibilities of developing a system enabling free head movement without tedious geometric calibration.

Although eye tracking research is active more than several decades, it is hard to refer to a breakthrough in the field. It is easy to see adequate evidences to confirm this by viewing two most comprehensive surveys on the topic published in 35 years apart [45, 22]. Although the recent developments in imaging and computing technologies, in both articles, majority of the methods have accuracies about 1° and the best reported accuracies are around 0.5°. Due to the inconstancy of physiological and physical structure of eyes, variability in lighting conditions, occlusions, and differences in scale and appearance; further improving the accuracy and stability of eye tracking systems still remains challenging.

Eye tracking algorithms consist of two main steps: feature extraction, and gaze estimation. In the first step necessary information such as face, eyes, center of pupil(s) and corneal reflection(s), etc. are detected with the camera(s) and/or other sensors. Feature extraction methods can roughly be classified as shape-based, feature-based and appearance-based methods.

In the second step, i.e. gaze estimation, location of gaze is computed by processing the information detected in the previous step. Employed gaze

Table 3.1: Pros & Cons of Gaze Estimation Approaches.

	2D REGRESSION-BASED METHODS	3D MODEL-BASED METHODS
Pros	✓ Implicitly model the optical and physiological properties of eye ✓ Do not require camera or geometry calibration, hence simple to implement ✓ Can operate with less amount of hardware (for no or little head movement)	✓ **Provide head pose invariance** ✓ Can detect 3D line of sight (LoS) ✓ Require personal calibration rarely
Cons	× **Do not provide head pose invariance** × Cannot detect line of sight (LoS) × May require recalibration frequently (before each use)	× Require camera and geometry calibration × Difficult to model human eye due to the complex structure × Require multiple glints to robust estimate of cornea center × Use extra camera and PTZ unit to enable large head movements × Require modeling the refraction for glasses and contact lenses

estimation method significantly affects the characteristics and performance of an eye tracker, hence, studies related to gaze estimation problem constitutes the major part of the literature.

In this study we leave the feature extraction step beyond the scope and concentrate on the gaze estimation problem. Gaze estimation algorithms can be listed in two main categorizations:

1. *2D Regression (Mapping) Based Methods:* In 2D regression-based methods [47]-[53], detected eye features (pupil center, corneal reflections (glints), etc.) are associated with gaze coordinates with a mapping function. The coefficients of this function is determined with a calibration process which relates the eye features and the gaze locations. Behaviour of the mapping function can be linear or nonlinear, however, a second order polynomial with cross terms is reported for giving the best results in most studies.

2. *3D Model (Geometric) Based Methods:* In 3D model-based methods [54]-[61], structure of the human eye is geometrically modelled to compute the 3D gaze vector orienting from the cornea center. Thus the gaze point is obtained by intersecting the gaze vector with the scene information. 3D model-based methods usually require two calibration schemes. First one is the geometric calibration of the camera(s) and environment, and is performed only once to obtain the necessary priori knowledge. The second one is personal calibration which is performed for each user to determine the parameters specific to human eye such as the cornea radius and angle between optical and visual axes.

Both 2D regression-based and 3D model-based methods (RBM and MBM in short) are thoroughly studied and many advantages and disadvantages are stated [22, 46]. However, the problem of head-pose invariance is still a tough problem for all types of eye trackers regardless the employed gaze estimation method.

In this thesis study we propose a hybrid gaze estimation model which aims to combine the advantages of both 2D regression-based and 3D model-based methods. In our study, we employ a head-mounted eye tracker consisting of two cameras, for viewing the eye and the scene concurrently. Therefore, we extract the pupil center from the high resolution images acquired from the eye camera, at the same time, we accurately estimate the head pose with the scene camera by the help of fiducial markers. After that, we fuse the information gathered from both cameras to estimate the PoG. Another contribution of the thesis is *hierarchical calibration* procedure which is an enhancement technique for the accuracy of mapping function.

The rest of this section is organized as follows. Advantages and disadvantages of the regression-based and model-based gaze estimation methods

are explained in Section 3.2. We introduce the proposed method with the underlying motives in Section 3.3. In Section 3.4 and Section 3.5 we explain the details of calibration and gaze estimation schemes for the proposed gaze estimation model. We present the experimental results of the proposed method in Section 3.6. Important aspects of the study are discussed in Section 3.7. Finally, we present the concluding remarks in Section 3.8.

3.2 Review of Gaze Estimation Methodologies

3.2.1 2D Regression-Based Methods

In 2D RBMs, relevant eye features are associated with gaze locations by a mapping function and, the coefficients of this function are obtained with a calibration procedure [47]-[53]. Once the coefficients are obtained, coordinates of eye features can be mapped to screen coordinates. However, the accuracy of this mapping decays in case of any head motion due to the displacements on eye features on the eye image.

To improve the invariance of head pose changes, the vector between the pupil (or iris) center and corneal reflection (PCCR) is employed [47, 50, 51, 52]. In this technique, the reflection of the light source has a relative displacement to head motions, thus the change on PCCR vector remains minimal. This technique can tolerate only small head movements and cannot compensate large pose changes. PCCR vector is also used in head-mounted eye trackers to compensate the slippage of the tracker.

2D RBMs implicitly model the eye with all non-linearities, as well as glasses and contact lenses if exist, hence they are easy to implement and use. However, those methods cannot detect the actual 3D line of sight (LoS) since they only associates the coordinates of eye image and screen. Consequently, they cannot provide significant head-pose invariance and they enforce users to keep their head still These kind of physical constraints dramatically reduces

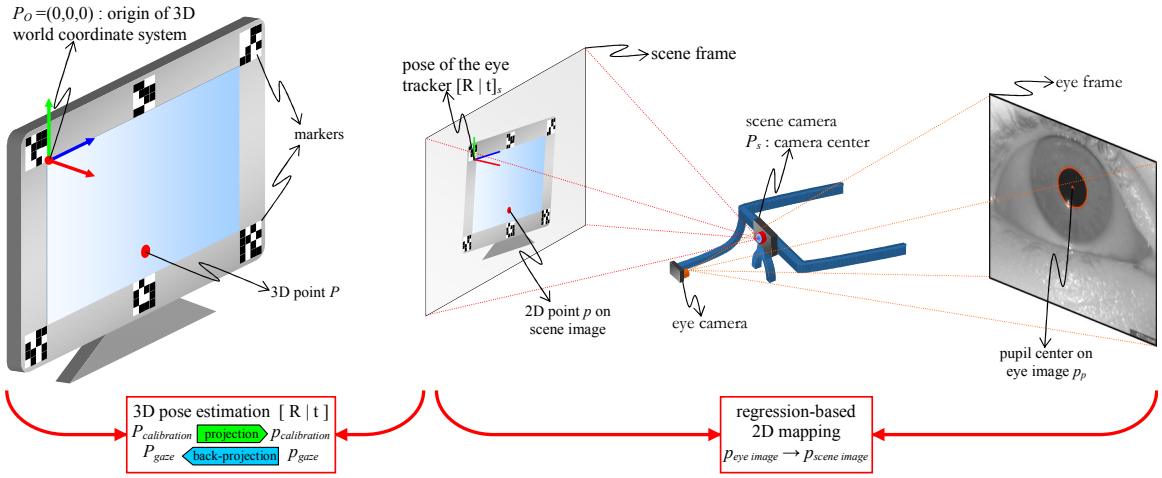


Figure 3.1: General model of the proposed system.

the comfort and ease of use especially for long sessions.

In [50], 2 cameras are employed to estimate the 3D head position and modify the regression function to compensate the head movements. Although this method relaxes the restrictions on head motion, it requires a prior stereo calibration opposite to other RBMs.

In addition to the video-oculographic gaze estimation systems, there are also systems which extracts features with simpler sensors instead of cameras [53]. Since the extracted feature information may not be sufficient to model the eye structure, RBMs are essentially more convenient for those systems. Eventually, these systems can easily map the extracted features to the screen locations and cannot provide head pose invariance without an additional mechanism.

3.2.2 3D Model-Based Methods

Model-based approaches are more successful on providing head-pose invariance, however, many methods require to be aware of the environment geometry to reconstruct the eye in 3D. They basically solve the head-pose invariance problem by estimating the gaze as a vector in 3D, for this reason, most of them require camera and geometry calibration in addition to the personal

calibration. Geometry calibration is the scheme that the 3D structure of the environment is modelled by measuring the positions and orientations of the employed components such as monitor, cameras, light sources, pan-tilt-zoom (PTZ) units, etc.

Most model-based methods assume that the cornea and eyeball has a perfect spherical surface. There are few number of studies that model these structures ellipsoids [54]. However, both approaches can fail due to the fact that a real human cornea may have a non-uniform curvature especially around the edges where the cornea merges with the limbus. Deficiencies on modelling the cornea can cause significant inaccuracies on detection of 3D glint locations. This situation causes errors on the detection of cornea center and results on a significant drift on the estimated gaze.

The most common feature extracted in MBMs is glint which is the reflection of a light source from the cornea surface. Glints are very helpful in estimation of various parameters of eye such as cornea radius and position of cornea center. Shih et al. [57] show that it is possible to detect the cornea center with one camera and two glints if the cornea curvature is known. However, it requires at least two cameras and two light sources when eye-specific parameters are unknown.

There are also studies that use population averages for human eye parameters (cornea curvature, refraction indices, angle between visual and optical axes, distance between pupil center and cornea center, etc.) overcome the problems related to geometrical modelling of human eye [55, 59].

Single camera systems are subjected to the tradeoff between high resolution eye images and tracking range. Otherwise, it requires to employ additional hardware (i.e., stereo cameras, PTZ units, light sources, etc.) to track eyes with high resolution in a wide area [56]. This situation complicates

the geometric calibration and causes the system to be error prone. Although, geometric calibration is performed only once (unless any change is applied to the components), it is a troublesome process that may require accurate measurements of geometry. In some studies mirrors are employed since PTZ units may decelerate the responsiveness of the tracker due to the mechanical operation [54, 59].

Guestrin and Eizenman [60] state the relations between the glints and gaze estimation in fully calibrated scenarios which require Euclidean information for the cameras, human specific parameters, light source positions, and camera parameters. Villanueva and Cabeza [61] prove that, regardless the number of light sources and cameras, at least one calibration point requires to estimate the angle between optical and visual axes in a MBM.

It is shown that increasing the number of employed glints improve the accuracy in both 2D RGM and 3D MBM approaches [22, 61], however, there are also several difficulties on using glints. First, using specular reflections makes working at outdoor difficult due to the uncontrolled lighting conditions. Second, if system uses multiple light sources, all glints have to appear for robust gaze estimation. Otherwise, accuracy might significantly reduces in case all of the glints cannot be detected due to the eyelid/eyelash occlusions, pose differences or reflections. Moreover, to get benefit from multiple glints, cornea has to be modelled very accurately due to the non-spherical geometry of it.

Besides its benefits, there are important issues that need to be considered about glints regarding approaches including techniques using PCCR vector. First, contrary to the assumption, glints displace as the eye moves since it rotates around the eyeball, not around the cornea center [22]. Therefore, positions of glints change in 3D with the whole cornea surface as well.

Second, the location of glints can be slightly effected due to the non-spherical curvature of cornea.

Another important point on 3D eye modelling is the effect of refraction to the glints and other eye features. Refraction causes features to be observed on different lines rather than they are actually located. Omitting or inefficiently modelling the refraction factor may perturb the gaze accuracy 1 degree or more. Therefore, there are studies attempt to reconstruct the pupil in 3D according to the refraction index to improve the accuracy [58, 61].

All the aforementioned issues related to the model-based approaches can considerably effect the estimated gaze location unless they are considered in a sensitive manner. Although regression-based methods implicitly handle most of these issues, head-pose invariance significantly hinders their practical use. In Table 3.1 we present a brief taxonomy regarding the advantages and disadvantages of the gaze estimation approaches.

3.3 Proposed Method

In this study, we propose a hybrid gaze estimation method that intends to merge the upsides of both RBM and MBM approaches. In our method, we aim to overcome the challenges related to 3D modelling of human eye by using a regression-based mechanism. At the same time, we provide free head movement by accurate 3D pose estimation with the help of fiducial markers. We extract the markers from the scene images and use 3D-2D correspondences to find the location and orientation of the apparatus in 3D precisely. In Fig. 3.1 we present the gaze estimation model of the proposed method. We map the eye image coordinates by a regression based method, and associate the scene image coordinates with real world by computed pose matrix.

We develop a lightweight goggles consist of two webcams, one for mon-

itoring the eye, and the other is for monitoring the scene (see Fig. 3.2). There are several reasons for preferring a head-mounted (HMD) eye tracker in this study. First, precision of feature extracting scheme deserves significant attention for a reasonable eye tracking accuracy. It is a fact that the more resolution the system acquire; the more precision and accuracy the eye tracking system provides. In a HMD system, it is easy to sample the eye in high resolution that is essential for precise feature extraction, hence accurate estimation.

Similarly, the accuracy of head-pose computation is crucial for efficient position compensation. There are remote gaze estimation studies try to estimate the head pose by face detection [64]. Yet, head pose estimation with this technique cannot provide sufficient accuracy due to the discrepancies of human facial features. For this reason, we incorporate a fiducial marker based pose estimation mechanism using the scene camera. In this way, we can estimate the head pose in millimeter precision.

Actually, using fiducial markers in gaze estimation applications is not new, however, they are usually employed in detecting the inter-frame correspondences rather than estimating the head pose. Then detected correspondences are utilized to merge the gaze information acquired during a session in a single frame to generate gaze analysis results like heat maps. For instance, in [63] markers are employed to find the homography among the frames for mapping the gaze locations to the reference frame.

As a complimentary motive for developing a HMD system, we can count the extreme large tracking regions that they provide. Even in the most sophisticated remote systems, users have to keep their heads in a predefined volume to get the best experience. In HMD systems this constraint can be removed, thus developing a diverse set of mobile applications becomes

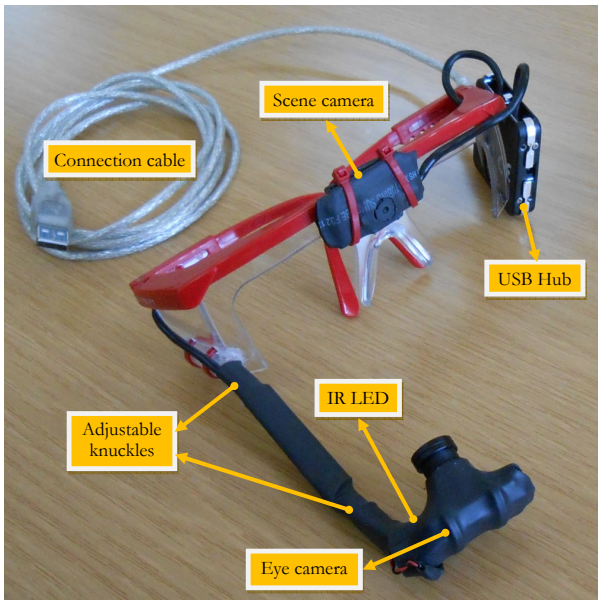


Figure 3.2: Apparatus built for experiments.

possible. Especially, multi-monitor scenarios are in our interest that two or more monitors can be utilized in the same system without an incrementation on the number of cameras.

We detect pupil contour and center with the method presented in the previous chapter whose article version is currently in revision [65]. We also estimate the pose with the help of EDMarkers library presented in [66]. The intrinsic calibration of scene camera is required for accurate estimation of head pose. On the contrary, intrinsic calibration of eye camera is not necessary because it is implicitly approximated during by the least square estimation based mapping operation. In Fig. 3.3 sample images acquired from eye and scene cameras are presented where the detected pupil contour and extracted markers with the pose information is shown. In the first image detected pupil contour, and in the second one, extracted markers and pose are shown.

Our method estimates the gaze in two steps. First, we find the gaze location on the image acquired by scene camera (scene frame in short). Con-

sidering the scene frame as a virtual display mounted to the head at a distance of focal length of the scene camera eases to explain the proposed gaze estimation methodology. Obviously, the gaze location on scene image does not change with head-pose changes due to the fact that the virtual display has the same displacement and rotation with user's head.

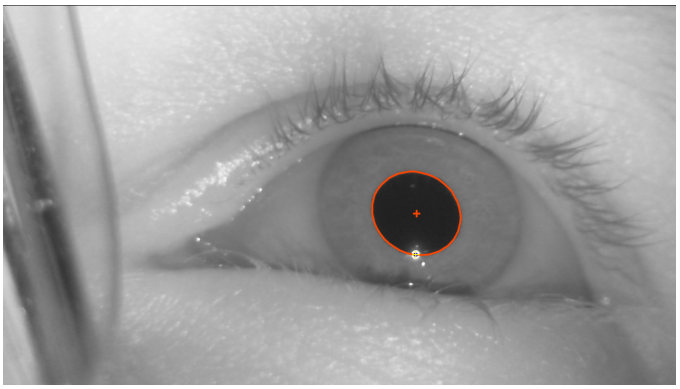
Apparently, finding the gaze on the virtual frame is easy and robust, however, it is not sufficient to find the actual gaze in 3D environment. To solve the tough part of the problem we use the head pose information that we obtain by markers. Head-pose information is rarely used explicitly in gaze estimation algorithms because there is no straightforward method to fuse regression parameters and head-pose information [22]. In this study we use common computational vision techniques to find the gaze point in real world. We give the details of the proposed system in the following sections.

3.4 Calibration

Mapping-based gaze estimation approaches work quite well unless head-pose invariance is the case. In these techniques, a mathematical relation is formulated between the known gaze locations and eye features (i.e., pupil center or PCCR vector). However, in our study, we aim to map the eye feature locations detected from eye camera to the certain locations on scene camera image. Technically speaking, we want to construct that mapping between the eye and scene frame coordinate systems. Therefore we have to use the projections of the 3D calibration points (CP in short) on the scene frame instead of the point locations itself.

Notation: Remember the projection of 3D points to an image plane in pinhole camera model [69] ;

$$\lambda \, \check{p}^T = \mathbf{K} \, [\mathbf{R} \mid \mathbf{t}] P^T \tag{3.1}$$



(a) Eye image in 1280×720 resolution.

(b) Scene image with dense markers in calibration mode.

Figure 3.3: Sample camera images acquired from eye and scene cameras, respectively.

where λ is the arbitrary scale factor in homogeneous coordinates, P is a 3D point in world coordinate system, $\check{p}$ is a 2D point which is the projection of P. $\mathbf{K}$ is intrinsic camera matrix, $\mathbf{R}$ is rotation matrix and $\mathbf{t}$ is translation vector of the camera, respectively. Notice that $[\mathbf{R} \,|\, \mathbf{t}]$ is called the extrinsic camera parameters and represents the pose of the camera. Herewith, we denote p as the undistorted image coordinates of $\check{p}$ and, for convenience, we denote $\mathbf{H}$ as the entire mathematical transformation (consisting of projection and distortion correction) applied to P such that $p = \mathbf{H}P$ where $p = (x, y)$ and $P = (X, Y, Z)$ are 2D and 3D points, respectively.

During calibration, we acquire frames from both eye and scene cameras synchronously as the user is looking at the predefined 3D calibration points appeared on the monitor. We obtain pupil center $p_p = (x_p, y_p)$ from

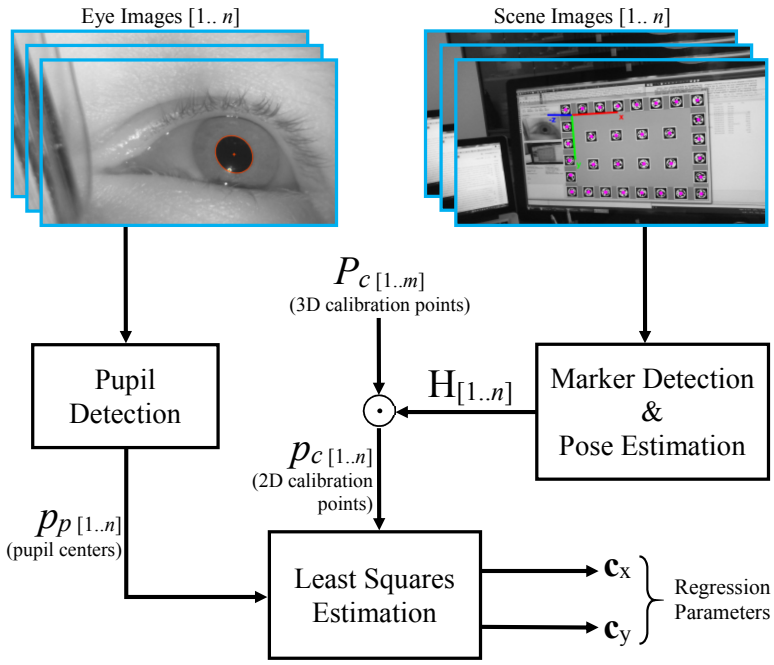


Figure 3.4: Calibration schema where the information gathered from scene and eye cameras is fused to obtain regression parameters.

eye camera and extract markers with EDMarkers library from scene camera frames (See Fig. 3.3). Since we need to compute extrinsic parameters with 3D-2D correspondences, we calibrate the scene camera beforehand to obtain the $\mathbf{K}$ matrix and distortion coefficients. With the detected markers, we compute the pose of the scene camera $[\mathbf{R} \mid \mathbf{t}]_s$ which also corresponds to the head-pose as well. For computation of $[\mathbf{R} \mid \mathbf{t}]_s$ we use the OpenCV's pose estimation function with the Levenberg-Marquardt algorithm as the optimization method [67, 68].

Once we gather the necessary information consisting of pupil centers and pose matrices from eye and scene cameras respectively, we can construct the mapping between eye frame and scene frame coordinate systems. To do this, we need to know the corresponding locations of the scene images for the 3D CPs displayed on the monitor. In our gaze estimation model (see Fig.3.1), we accept the upper left corner of the monitor as the origin of the world coordinate system $P_O = (X_O, Y_O, Z_O)$. In this scenario, assume that

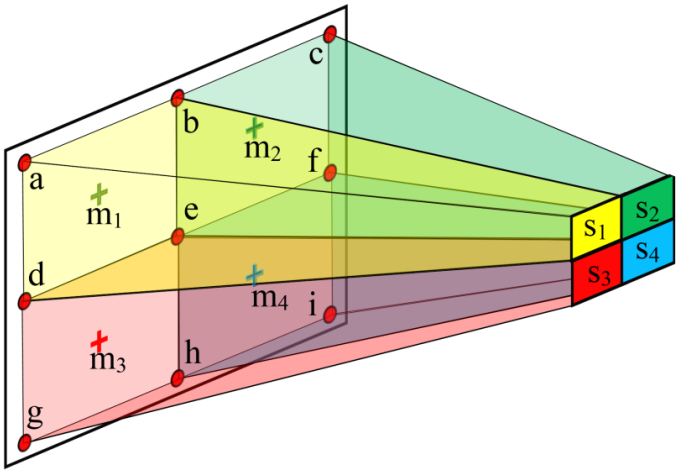


Figure 3.5: *Hierarchical calibration:* We divide screen into smaller regions and recalibrate each sub-screen ($s_{1..4}$) to enhance the accuracy of gaze estimation.

the user is looking at the i^{th} of the m CPs $P_{c_i} = (X_{c_i}, Y_{c_i}, Z_{c_i})$ displayed on the monitor; we can compute the location of the projection of that CP as the following;

$$p_c = \mathbf{H}_s P_c \tag{3.2}$$

where $\mathbf{H}_s$ is transformation matrix of scene camera, P_c is 3D CP displayed on monitor in world coordinates and p_c is 2D CP in scene image coordinates.

For each scene frame containing p_c as the projection of a certain 3D CP, we also have a concurrent eye image with a pupil center p_p. To form an equation system for $p_p \longrightarrow p_c$ mapping with eye and scene image pairs, we need to generate a feature polynomial to find the mapping coefficients. Employing higher order polynomials has been shown to improve the accuracy of regression-based approaches, however, increasing the number of terms requires more CPs in the end. As a result, it is found that the second order polynomials offers a good compromise between the number of CPs and the gaze estimation accuracy.

After the acquisition of n scene and eye image pairs during a calibration scheme, we set the following equation system to find the mapping coefficients

with least squares estimation:

$$\begin{bmatrix} 1 & x_{p_1} & y_{p_1} & x_{p_1}y_{p_1} & x_{p_1}^2 & y_{p_1}^2 \\ 1 & x_{p_2} & y_{p_2} & x_{p_2}y_{p_2} & x_{p_2}^2 & y_{p_2}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{p_n} & y_{p_n} & x_{p_n}y_{p_n} & x_{p_n}^2 & y_{p_n}^2 \end{bmatrix} \begin{bmatrix} c_{x_0} \\ c_{x_1} \\ \vdots \\ c_{x_5} \end{bmatrix} = \begin{bmatrix} x_{c_1} \\ x_{c_2} \\ \vdots \\ x_{c_n} \end{bmatrix} \tag{3.3}$$

where $p_{p_i} = (x_{p_i}, y_{p_i})$ and x_{c_i} are the coordinates of pupil center and x component of 2D CP p_c, respectively, both belong to the i^{th} scene - eye image pair. $\mathbf{c}_x = [c_{x_0}..c_{x_5}]^T$ is the 6-dimensional coefficient vector which maps any p_p to a corresponding x component of scene image coordinate system. Similarly, the same system is also established and solved with y components of 2D CPs (p_c) to find the $\mathbf{c}_y$ vector which maps the detected pupil center to the y component of scene image coordinates. In Fig. 3.4 we present a block diagram explaining the proposed calibration method.

3.4.1 Hierarchical Calibration

In regression-based methods relevant features of eye image such as pupil center are utilized to map the screen coordinates. When a user is gazing the screen from one edge to the other, displacement of the pupil center on the image forms a nonlinear trajectory. The characteristic of this trajectory is strictly depends on the camera's location and view angle while monitoring the eye. For instance, if the camera is monitoring the eye from a lower level, vertical movements of the pupil center becomes less perceptible due to the spherical structure of eyeball.

Viewing angle of the camera and spherical trajectory of pupil causes the mapping function to operate in a nonuniform manner. In other words, mapping function provides different accuracies on different portions of the screen because the obtained mapping coefficients are computed for whole

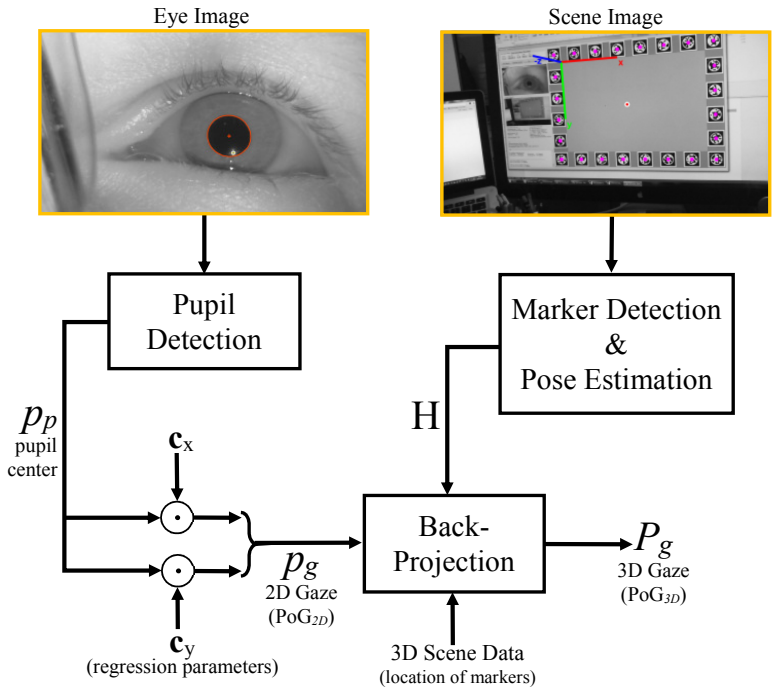


Figure 3.6: Gaze estimation schema. Since we define the 3D scene with markers as a surface representing the monitor, we do not need to provide additional data.

screen in a global manner. To reduce this deficiency of regression-based gaze estimation and enhance the accuracy; we propose a simple and efficient two-level calibration scheme that we term *hierarchical calibration*.

In the first step of this scheme we perform global calibration which covers the whole screen just same as the conventional methods. In the second step, we perform individual calibrations for certain portions of the screen. In this manner, regression parameters can fit the smaller portions of the screen instead of the whole screen. Therefore, nonuniformity of the mapping function can be modelled better and smaller errors can be obtained.

In Fig. 3.5 we show an illustration as an example of hierarchical calibration for nine point calibration scheme. In the first step, we calibrate the system for all CPs (a to i) to find $\mathbf{c}_x$ and $\mathbf{c}_y$ vectors which map any eye feature vector to screen coordinates. In the second step, we perform calibration operations for all subscreens (s_1 to s_4) with four CPs for each and find local

regression parameters. For example, in the case of s_1, we find $\mathbf{c}_{xs_1}$ and $\mathbf{c}_{ys_1}$ with the data of {a, b, d, e} CPs.

During the gaze estimation, we first find the gaze point with global coefficient vectors $\mathbf{c}_x$ and $\mathbf{c}_y$ to determine the gaze point on the screen. At the second step, to find the subscreen which the gaze falls on, we compute the Euclidean distances between the gaze and subscreen centers (m_1 to m_4 in Fig. 3.5). After the subscreen is determined by selecting the shortest distance, enhanced gaze location is computed by multiplying the eye feature vector with the corresponding subscreen's regression parameters.

3.5 Gaze Estimation

The calibration procedure is completed with the computation of regression parameters which maps the eye image coordinates to scene images coordinates. In gaze estimation step, we perform the identical operation in reverse order to detect the gaze in 3D. We obtain the 3D gaze in two steps; first, we find the 2D PoG (p_g) which is the gaze on the scene image, and second, we compute 3D PoG (P_g) location whose projection with the current pose information corresponds to p_g (refer to Fig. 3.1). We are giving the details for each step in the following subsections.

3.5.1 2D PoG Estimation

In the first step of gaze estimation scheme, we detect the gaze location on the scene image (p_g) with the regression parameters as the following:

$$p_{g_x} = c_{x_0} + c_{x_1}x_p + c_{x_2}y_p + c_{x_3}x_p y_p + c_{x_4}x_p^2 + c_{x_5}y_p^2 \tag{3.4}$$

$$p_{g_y} = c_{y_0} + c_{y_1}x_p + c_{y_2}y_p + c_{y_3}x_p y_p + c_{y_4}x_p^2 + c_{y_5}y_p^2 \tag{3.5}$$

where $p_p = (x_p, y_p)$ is pupil center detected from eye camera, $p_g = (x_g, y_g)$ is estimated PoG on scene frame (PoG_{2D}), and c_x and c_y are the coefficient

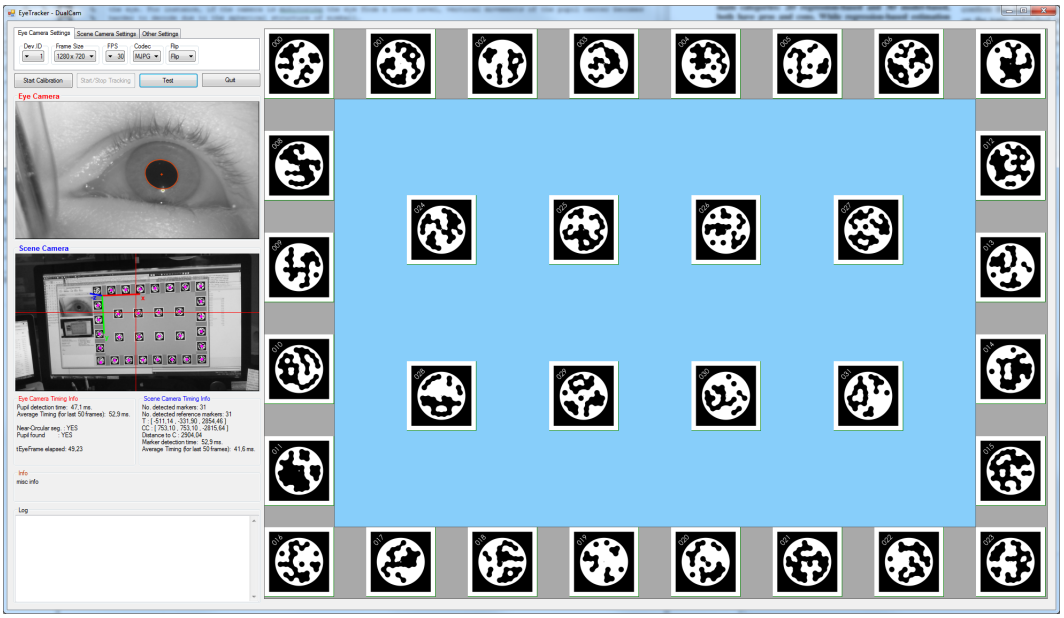


Figure 3.7: Snapshot of the software developed for the experiments. Streams of eye and scene cameras are displayed on the left. Blue region is gaze area in 1200×800 pixels resolution. Upper left corner of the gaze area is origin for both 2D gaze area and 3D world coordinates. Markers within the gaze area displayed only during the calibration scheme to improve the pose estimation accuracy.

vectors obtained with least square estimation. To find the enhanced PoG, we compute the distances between detected (p_{g_x}, p_{g_y}) and centers of the subscreens (see Fig. 3.5). In this way, we find the subscreen which the PoG falls. Afterwards we update the p_g by multiplicating the same feature vector generated from pupil center by the local regression parameters belonging to the corresponding subscreen. Obviously, PoG_{2D} is invariant to head-pose changes because the scene camera moves in the same direction and orientation with the head.

3.5.2 3D PoG Estimation

Detecting the gaze point on the scene image is the first part of the two step 3D gaze estimation. In the second step, we need to associate the 2D gaze coordinates on the scene image with the 3D world coordinates with the utilization of transformation matrix of the scene camera $\mathbf{H}_s$. In other words,

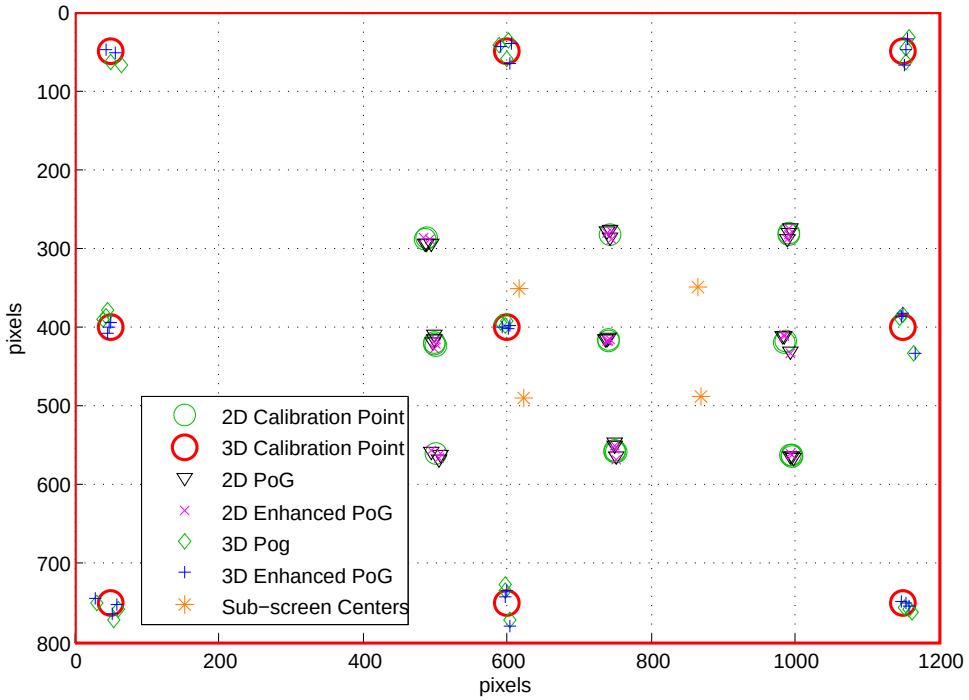
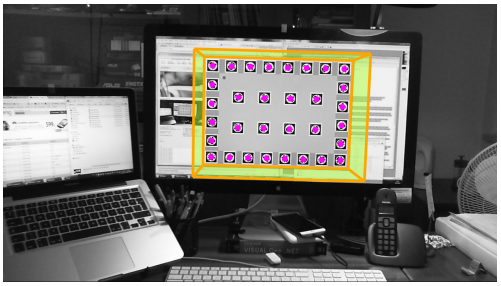


Figure 3.8: An example of calibration result.

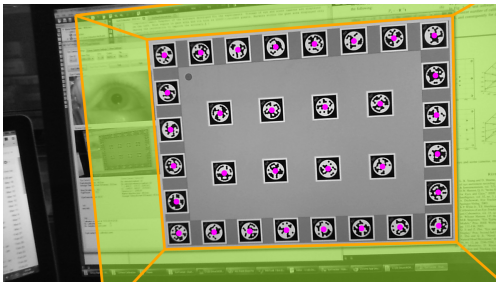
we need to find the 3D point P_g whose projection coincides with the 2D PoG (p_g) with respect to the given intrinsic and extrinsic parameters $\mathbf{K}\,[\mathbf{R} \mid \mathbf{t}]$. This operation is actually a back-projection and can be performed in a couple of simple steps when the pose information is available.

For an image point and a certain transformation matrix, there are infinite number of 3D points which are projected to the same 2D point. Therefore, back-projection of a 2D point to 3D corresponds to a ray, obviously. To find the actual gaze point, we need to intersect that ray with the 3D scene information. Notice that 3D information in our case is surface of a monitor, thus it is a plane with a simple 3D structure. In our system configuration, we set the upper left corner of the monitor as system origin P_O, hence Z components of all points on the monitor are zero (refer to Fig. 3.1 and Fig. 3.7).

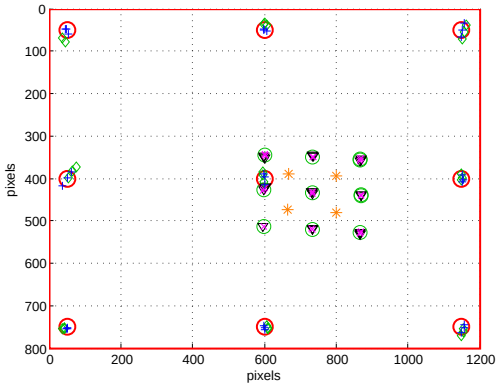
To find the ray equation, required information consists of the intrinsic ($\mathbf{K}$) and extrinsic ($[\mathbf{R} \mid \mathbf{t}]$) parameters of the camera. Since the ray intersects



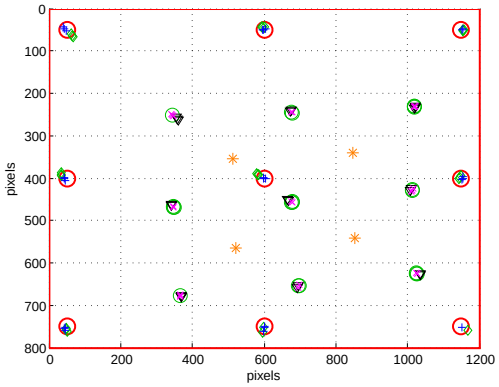
(a) Scene camera image from long distance.



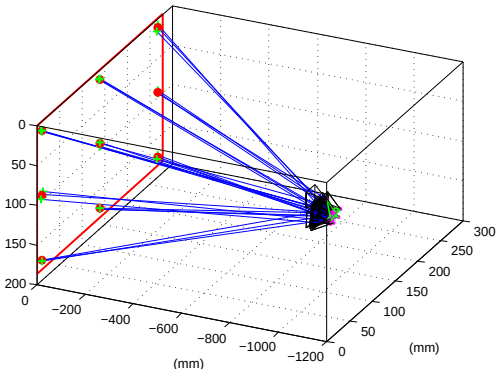
(b) Scene camera image from short distance.



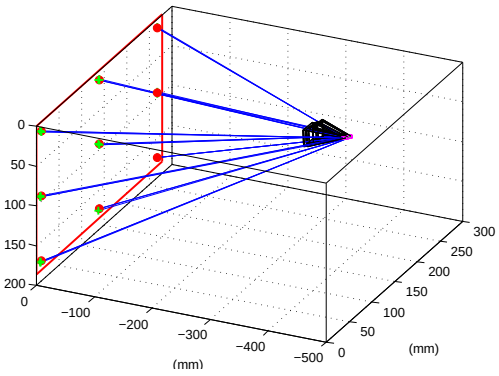
(c) 2D PoG error sketch.



(d) 2D PoG error sketch.



(e) 3D Calibration result.



(f) 3D Calibration result.

Figure 3.9: Calibration results for two sessions from different distances. (a, c, e) from ~45 cm, (b, d, f) from ~110 cm.

the camera center, first the location of the camera center in world coordinate system needs to be computed. In our scenario we need to compute the center of scene camera with the extrinsic parameters as the following:

$$P_s = -\mathbf{R}^{-1}\mathbf{t} \tag{3.6}$$

where $P_s = (X_s, Y_s, Z_s)$ is the center of scene camera in world the coordinate system originated with upper left corner of gaze area shown in Fig. 3.7. Once the camera center P_s is found, we associate it with the pixel coordinates of the image point which is also coincides with the same ray:

$$L(\lambda) = P_s + \lambda \mathbf{R}^{-1}\mathbf{K}^{-1}p_g \tag{3.7}$$

where λ is the scaling factor and the $L(\lambda)$ is a function of λ and represents the point set which constitutes the ray [69]. Obviously, the point that we are looking for is the one among the set of points identified by $L(\lambda)$. For a known depth Z, it is possible to find the X and Y coordinates in 3-dimensions:

$$\lambda = \frac{Z - Z_{P_s}}{v_3} \tag{3.8}$$

where Z_{P_s} is the Z component of the scene camera center and $(v_1, v_2, v_3)^T = \mathbf{R}^{-1}\mathbf{K}^{-1}p_g$.

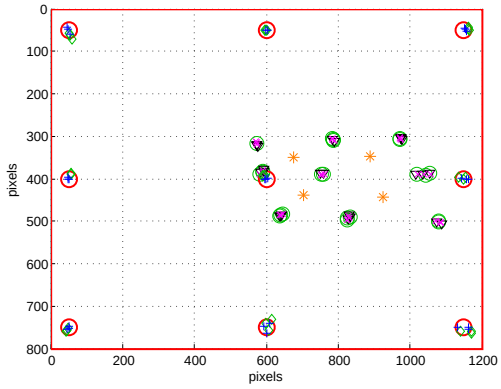
We define the 3D structure of our system with the help of markers displayed on the monitor, thus we do not need to perform tedious geometric calibration. We display the markers in the specific locations and assume that the surface of the monitor corresponds to $X - Y$ plane. In this way Z components of all points on the monitor becomes zero and we easily find PoG_{3D} by setting $Z = 0$ to find the gaze point in Eq. 3.8.

3.6 Experimental Results

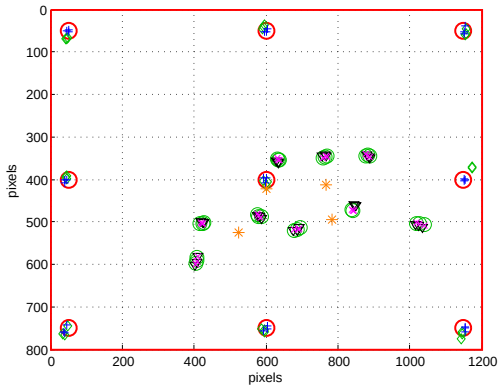
In this section we present the performance of the proposed method in terms of accuracy and stability. Since the proposed algorithm estimates the gaze in two steps and the efficiency of the obtained gaze strictly depends on the success of both. For example, if the detection of PoG_{2D} is not accurate, than the obtained PoG_{3D} by back-projection ends up inaccurate as well. For this reason, we present results in both 2D and 3D in terms of accuracy. In Fig. 3.7 the user interface that we develop for experimental issues is shown. We perform experiments with the apparatus that we build with two webcams (Microsoft HD 3000) on a safety glass (see Fig. 3.2). We only modified the eye camera by mounting an 8 mm lens and IR-pass filter. We can deliver 1280×720 images synchronously from both cameras.

In Fig. 3.8 we present a calibration result with legend for a nine-point calibration scheme. We explain how the sketches of calibration results should be interpreted by the reader on this figure. We superimpose the 3D and 2D PoG sketches to show the relation between 3D and 2D gaze estimation accuracies and effect of the calibration distance. In the sketch, 3D CPs correspond to the CPs displayed on the monitor and 2D CPs are their projection to the scene image with the pose which the image pair is sampled with. 2D PoG and 2D Enhanced PoG are gaze locations on the scene image computed by using the global and local (obtained by hierarchical calibration) regression parameters, respectively. Sub-screen centers are the centroids of the corresponding 2D CPs.

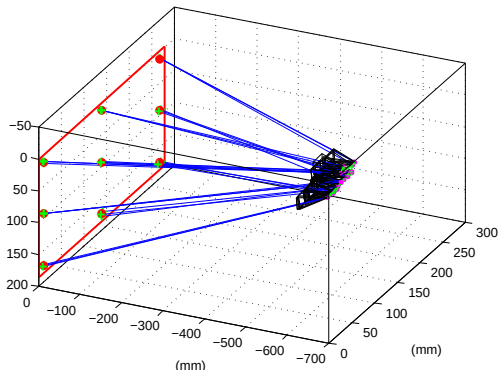
In the experiments we sample 3 image pairs for each CP, therefore, for nine-point calibration scheme we sample $9 \times 3 = 27$ eye-scene image pairs and it takes about 25 seconds in total. If the user moves his/her head, 2D CPs will not overlap and scatter according to pose changes for each eye-scene



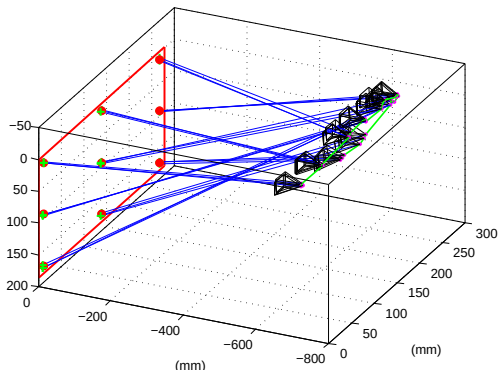
(a) 2D calibration result with small head motion.



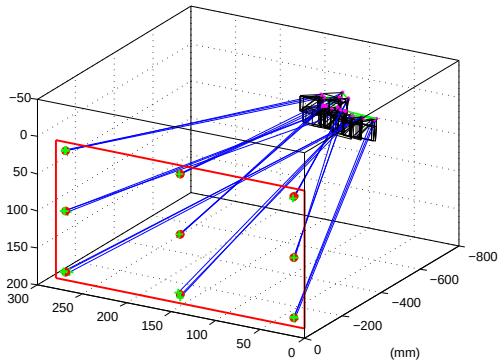
(b) 2D calibration result with large head motion.



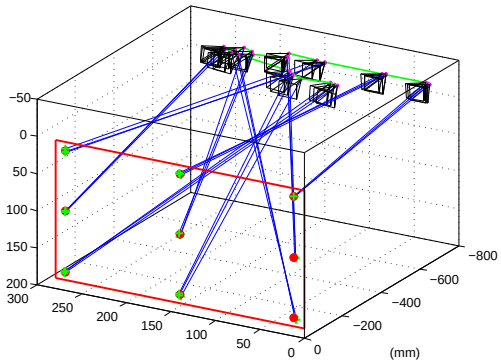
(c) 3D view of (a).



(d) 3D view of (b).



(e) Another 3D view of (a).



(f) Another 3D view of (b).

Figure 3.10: Calibration results with different amount of head motions from ~70 cm distance. (a, c, e) with small head motion. (b, d, f) with large head motion.

image pair (see Eq. 3.2). 3D PoG and 3D Enhanced PoG are the back-projected gaze locations in world coordinate systems for 2D PoG and 2D Enhanced PoG, respectively. From the sketches it is clearly seen that the enhanced PoGs reach the target CPs more accurately. The legend presented in Fig. 3.8 is valid for all other experimental results in the paper, hence we will not give any other legend to save space. In 3D box sketches (Fig. 3.9.c & f) we show the 3D CPs (red dots) and pose of the eye tracker with respect to the monitor in 3D to visualize the head displacement and pose change. In this sketch, we also display the back-projected rays (blue) and 3D enhanced PoGs (green plus signs).

In Fig. 3.9 we present the data for two calibrations, from longer (Fig. 3.9.(a-c)) and shorter (Fig. 3.9.(d-f) distances, respectively. Notice that during the calibration from the longer distance, eye movements remain minimal because the CPs are viewed closer to each other as the user move away from the monitor. As a natural consequence of this, the region which 2D CPs cover becomes narrower with respect to the calibration distance. This is because of the scaling effect of the projective transformation which causes objects to be observed smaller as the distance increases. Eventually, saliency of the user's eye movements decreases as the distance to the monitor increases, hence the precision of the gaze estimation diminishes.

In Fig. 3.10 we present the experimental results for two calibration schemes for small (Fig. 3.10.a-c) and large (Fig. 3.10.d-f) head motion. It is clearly visible from the 2D PoG sketches in Fig. 3.10.a & d, locations of 2D CPs are effected with respect to the scene camera pose variations due to the fact that 2D CPs are projections of 3D CPs. In this manner pupil centers are associated with the projections of 3D CPs and head motions can be compensated even during the calibration scheme. This mechanism can compensate medium level head motions very effectively, however, the efficiency of the

calibration degrades for very large head motions. In large head motions 2D CPs are scattered in an uncontrolled manner, and consequently, success of the hierarchical calibration decreases. To maximize the robustness and accuracy of a calibration, homogeneous distribution of 2D CPs, hence sub-screen centers is essential. Even though the proposed algorithm can compensate minor head movements during calibration, the user should keep his/her head still for a better accuracy.

3.6.1 Gaze Estimation Accuracy

Similar to many existing methods, we compute the gaze error in angular form. Since we compute the 3D pose of the eye tracker, we can accurately measure the distance between the eye tracker and any 3D point, as well as the points on the screen. In this way, we can accurately calculate the angular gaze error for each individual eye-scene image pair after the completion of calibration. We compute the amount of gaze drifts for 2D & 3D PoGs then we evaluate angular shifts. In most studies, angular accuracies are computed in horizontal and vertical separately. However, measuring accuracies separately becomes meaningless for head-pose invariant systems since head may also perform roll action. In that situation vertical movements of eye both have vertical and horizontal components on the target plane. Therefore we provide all results in magnitude form which is the gaze error in terms of degrees.

We run experiments with 10 subjects multiple times for each, and take the median accuracy values by eliminating best and worst performances. In Fig. 3.11 we present the gaze accuracies that we measure during the experiments. It is expected to have lower PoG_{3D} accuracies since the errors are being accumulated during the two step 3D gaze estimation scheme. The reason behind the PoG_{2D} error is the inefficiency of the mapping function. After estimation of PoG_{2D} we back-project the p_g and intersect it with the

Table 3.2: Average gaze estimation accuracies.

Accuracies (°)	PoG_{2D}	Enhanced PoG_{2D}	PoG_{3D}	Enhanced PoG_{3D}
Minimum	0.08	0.04	0.23	0.12
Maximum	0.62	0.46	1.19	0.89
Average	0.23	0.15	0.55	0.37

monitor plane to obtain PoG_{3D}. In this step gaze estimation error increases due to the errors accumulated from intrinsic camera calibration and pose estimation schemes. In Table 3.2 average accuracy values are shown for the 2D and 3D PoG estimations. The effect of the hierarchical calibration can be seen from the results which improves the accuracies up to %36 and %33 for 2D and 3D PoG estimation, respectively.

3.6.2 System Performance

We can acquire HD frames in 1280×720 pixels resolution from both eye and scene cameras simultaneously. Processing time of one eye image to extract pupil center takes ~20 ms on an Intel Core i7 2.80 GHz CPU. For scene images, it takes about ~45 ms to extract the markers and compute the transformation matrix on the same machine. Combining the information gathered from eye and scene cameras to compute the PoG_{2D} and PoG_{3D} takes negligible time. Consequently, whole gaze estimation scheme for one eye and scene image pair takes about 65 ms. Therefore proposed system can run in ~15 Hz for the certain implementation. In addition, processing of eye and scene images are independent and thus they can be executed in separate threads in parallel by utilizing the multi-core CPUs. In this manner real-time performance of the system can be improved. Along with the results given in the paper, we further present demo videos and other supplemental materials in our web page [70].

3.7 Discussion

The proposed gaze estimation model easy to deploy and requires only personal calibration for the computation of regression parameters. Eye and scene cameras do not require a joint calibration since their field of views do not intersect. However, scene camera requires the computation of intrinsic parameters which is an operation performed only once. Before personal calibration, user can adjust the eye camera with the knuckles on the camera's arm in order to acquire the best eye images (see Fig. 3.2). The physical changes that the user applies to the goggles are compensated with the personal calibration. The system does not require geometrical calibration since we infer the 3D structure by the help of markers and we display them on the predefined coordinates of the monitor. Using markers may reduce the effective area of the monitor, but, this drawback can be solved easily by hiding markers around the computed gaze location in application.

As an important feature, proposed system can be extended for multi-monitor systems very easily by defining each monitor as an individual 3D plane and displaying different markers. It is not necessary to measure the relative positions of the monitors since the pose is computed automatically for each monitor.

Another critical advantage of the proposed system is the capability to work with glasses and lenses. Because we associate the eye features with the scene image coordinates by a mapping function, we indirectly model the glasses and lenses if any exists.

There are couple of deficiencies of the proposed system. First, we cannot compute the actual line-of-sight (LoS) like the majority of the model-based approaches. The difference is obviously the origin of the gaze vector. The origin of the computed gaze vector is the cornea center in existing MBMs,

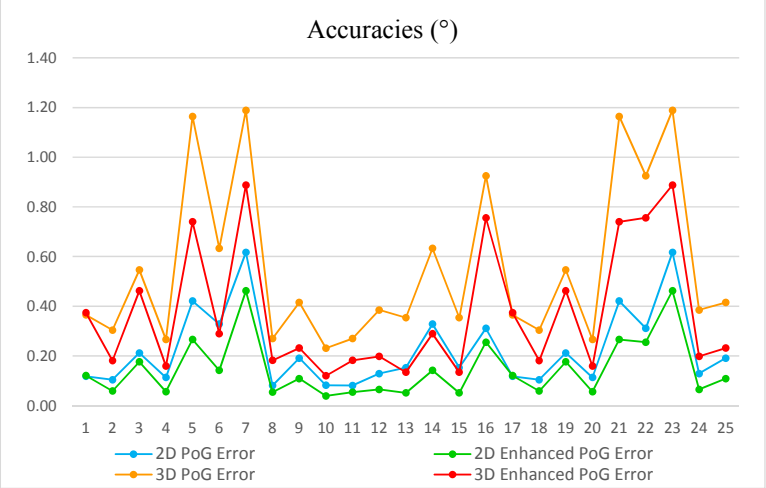


Figure 3.11: Gaze accuracies for PoG_{2D}, PoG_{3D}, Enhanced PoG_{2D}, Enhanced PoG_{3D}.

however, it is principal point of the scene camera in our approach. Actually, we do not know any specific gaze estimation application which suffers from this situation.

Second, similar to all head-mounts, our system suffers from slippage, and therefore, efficiency of the calibration decays. To reduce the effect of slippage we also test our system with using PCCR vector instead of pupil center. In the experiments, we see that the effectiveness of the calibration remains longer when the PCCR vector is employed. However, PCCR vector cannot properly work in hierarchical calibration and thus results in a lower accuracy compare to the pupil center.

Another problematic coming with the head mounted eye trackers is parallax error [72]. It becomes very prominent especially in mobile eye tracking applications where distances of fixation points significantly vary [74]. In our experiments, effect of the parallax error becomes noticable when the user dramatically changes the distance to the target plane. There are couple of methods for parallax error compensation in the literature, i.e. optical [71] and computational [73].

3.8 Conclusions

In this chapter of the thesis we propose a gaze estimation model which aims to combine the advantageous sides of both 2D regression-based and 3D model-based gaze estimation approaches. Proposed system is able to detect 3D PoG without requiring tedious geometric calibration. We obtain gladsome experimental results by computing the head pose very accurately and modeling the 3D of monitor precisely without any hand measurements, both with the help of markers. As a unique property, our gaze estimation approach can even compensate the small head movements occurred during the calibration and perform with a reasonable accuracy. By the help of fiducial markers, it is possible to use even multiple displays without calibrating their positions with respect to each other. Markers with different IDs can be placed on different displays and hence they have individual coordinate systems. The scene camera can distinguish the display which the user is looking at and estimate the pose. In this way gaze of the user can be estimated. Multiple display scenario is a tough situation for remote systems due to the narrow tracking range even though they are supported with a PTZ unit.

In the literature remote systems constitutes the major part on the contrary to the head-mounted systems. However, increased interest to the mobile eye tracking, wearable computers and personal displays can increase the demand for head-mounted eye tracking in near future. In this way, head-mounted eye trackers can lead up to hands-free interaction with mobile devices because of the reliability against external factors they provide. Furthermore, the association of gaze with the real world can lead up to many new interaction styles for the user with the environment they live.

In this work we only perform gaze estimation experiments which aim to find the gaze on a plane. For this task, we do not need to measure the

Table 3.3: Comparison of gaze estimation studies.

Method	Cameras / Lights	Head Pose	Calibration	Accuracy (°)	References
Regression Based	1 / 0	×	Personal	2 - 4	[87]
	1 / 1	×	Personal	1 - 2	[47, 49, 51]
	1 / 1	×	Full	0.75	[52]
Model Based	1 / 1	✓	Personal	0.5 - 1	[55]
	1 / 4	✓	Full	~1	[61]
	2 / 1	✓	Full	3	[59]
	2 / 2	✓	Full	< 1 - 2	[57, 58]
	3 / 1	✓	Full	0.7 - 1	[56]
	4 / 2	✓	Full	0.6	[54]
Hybrid	2 / (1)	✓	Personal	~0.35	Proposed

3D structure to apply geometry calibration thanks to the fiducial markers. Nevertheless, with the proposed method, it is also possible to find the gaze on an arbitrary object as long as the 3D model of the scene or object is available. For instance, gaze information of a user on a sculpture is also possible if the 3D model data of that sculpture is available. However, in this scenario, geometry calibration is required to associate the specific locations of the sculpture with the markers.

4. ON-SCREEN KEYBOARD: SliceType

4.1 Introduction

Once we can obtain reliable eye features in chapter 2 and compute the PoG with those features in chapter 3 we need to utilize this information in an application which is developed specifically to be used with gaze information. In this chapter of the thesis, we aim to design and develop an on-screen keyboard to enable boosting the text input throughput with plain gaze information.

Historically, the human-computer input device has been the keyboard before mouse was introduced in late 60s. People use keyboards of different layouts and functionalities to edit documents, write e-mails and short messages, prepare presentations etc. Although the physical keyboard remains to be the central text entry device for many people, it is not a viable solution for many people with disabilities.

To enable text input entry for individuals with disabilities who cannot use a traditional physical keyboard, On-Screen Keyboards (OSK) having different interfaces and functionalities have been designed over the years [75, 76, 77, 78, 79]. The main factor that determines the design of such keyboards is how the user selects an item, e.g., a button, a letter etc., from the OSK graphical user interface (GUI). In case the person has some form of motor ability in his/her voluntary muscles, one approach is to use a specialized switch for item selection. If the person is completely paralyzed and is unable to move any voluntary muscles except the eyes, for example those suffering from cerebral palsy (CP), amyotrophic lateral sclerosis (ALS) also known as Lou Gehrig's disease, or locked-in syndrome (LIS), eyes are the person's only input modality and interaction medium. In such cases, eye tracking and

Figure 4.1: Scanning keyboard in Windows 7. The current scan is at row 3. Letter '**k**' has been selected in the previous scan; therefore, the keyboard displays word suggestions at the first row.

blink detection are the only gestures for user input. Using one or both of these input modalities, different kinds of OSK designs have been proposed in the literature.

In this paper, we present a new OSK, named SliceType because of the shape of the character buttons, to help disabled individuals for fast text entry. SliceType can be operated by a pointing device such as a mouse or by eye tracking software that translates eye movements to mouse movements. Unlike most rectangular OSKs found in the literature, SliceType has a circular shape to minimize the distance between any two characters. Character frequency statistics of a corpus is utilized to lay out the characters on the OSK GUI. Users select a character by dwelling upon the button representing the character for the entire dwell period. As characters are selected, characters that may not follow the currently entered word prefix get removed from SliceType GUI and their button areas are merged with those of neighboring characters; thus making it possible to select the remaining characters more easily and accurately. To make typing faster, SliceType also starts presenting word suggestions within the same button area representing the current character (instead of using a separate area as is done by many OSKs) based on the entered word prefix, which the user may select by dwelling on the suggested word one more time. Furthermore, SliceType is designed to make word predictions based on the previous two words entered by the user, which

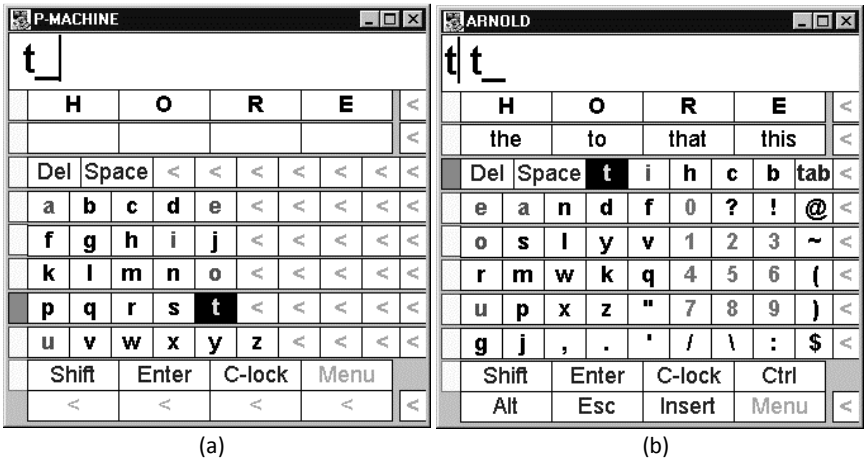


Figure 4.2: SoftType's user interface. Letter '`t`' has been selected in the previous scan. (a) The keyboard makes suggestions for the next letter, (b) Suggested words are displayed in the second row.

improves the quality of the word suggestions. Experiments performed by a set of novice users show an average text entry throughput of 5.2 words per minute (wpm) using a physical mouse as the input device, and 3.45 wpm using an eye tracker with a dwell period of 1000 ms.

4.2 Related Work

Existing OSKs found in the literature can be grouped in two general classes: (1) those that get operated by selecting a periodically highlighted item, e.g., a button, a pie slice, or any other GUI widget that can be selected. The item selection can be performed by a special physical switch if the user can operate certain voluntary muscles, or by eye blinking if the user is severely disabled and can only move eyes. (2) those that get operated by the movements of the computer mouse, either by moving the mouse over the appropriate item of the OSK GUI and dwelling on it for a period of time for selection, or by moving the mouse in certain paths to select certain items or to draw certain patterns. Since most such systems make use of an eye tracking software to translate the user's eye movements to mouse movements, they are usually called eye typing systems.

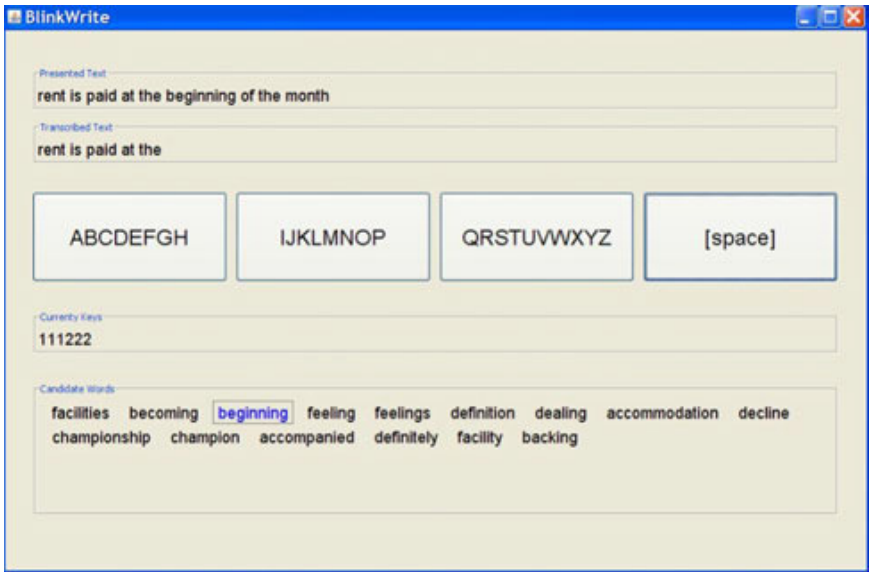


Figure 4.3: BlinkWrite's user interface.

OSKs that fall in the first category are generally called scanning keyboards. The idea with a scanning keyboard is very simple: The letters of the alphabet are laid out in some order on the OSK GUI. Each letter or a group of letters is periodically highlighted for a period of time before the scan moves to the next item. If the user wants to select the currently highlighted item, s/he uses a physical switch or uses eye blinks/winks to indicate the selection.

A good example for a scanning keyboard is the OSK that comes with standard Windows operating systems. Fig. 4.1 shows the scanning keyboard distributed with Windows 7. Instead of scanning over each letter, the Windows OSK performs row/column scanning for optimization. The scan starts at row 1. If the user wants to select a key from the currently highlighted row, s/he presses the space bar from the physical keyboard or uses some other physical switch device such as a joystick. A column-wise scan of the keys from the currently selected row then starts from left to right. To minimize the number of selections for a key, called the scan steps for character (SPS), the horizontal scan walks over the keys in the same row in groups of three

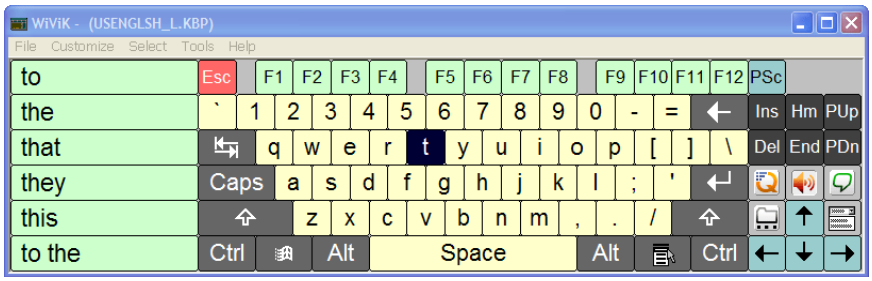


Figure 4.4: WiVik's rectangular user interfaces. There is a button for each character.

keys. Once the desired group is highlighted and the user selects the current group, a final scan over the individual keys within the group is performed key by key. The user finally selects the desired key after it is highlighted. It should be clear that it takes three user inputs to select a single key from this keyboard. Windows OSK also implements word completion as shown in Fig. 4.1. After the certain prefix of a word is entered, suggested words are displayed on the top-most row, which the user can select once highlighted. For example, in Fig. 4.1, the current scan is at row 3. Letter '`k`' has been selected in the previous scan; therefore, the keyboard displays word suggestions starting with letter '`k`' at the first row ordered with respect to some criterion such the frequency of occurrence in a corpus.

SoftType [80] is one of the earliest scanning keyboards with character and word completion capabilities. Fig. 4.2 shows SoftType's user interface, where letter '`t`' has been selected in the previous scan, which appears at the topmost textbox. The keyboard now makes suggestions for the next letter that may follow '`t`' by using the frequency statistics of the English language, which are displayed in the first row. The keyboard also starts making word suggestions in the second row. Authors state that the users make selections from the keyboard either by moving their head against one of three switches mounted in their chair, or by hitting a large banger switch [80].

Instead of using a physical switch for key selection, which cannot be uti-

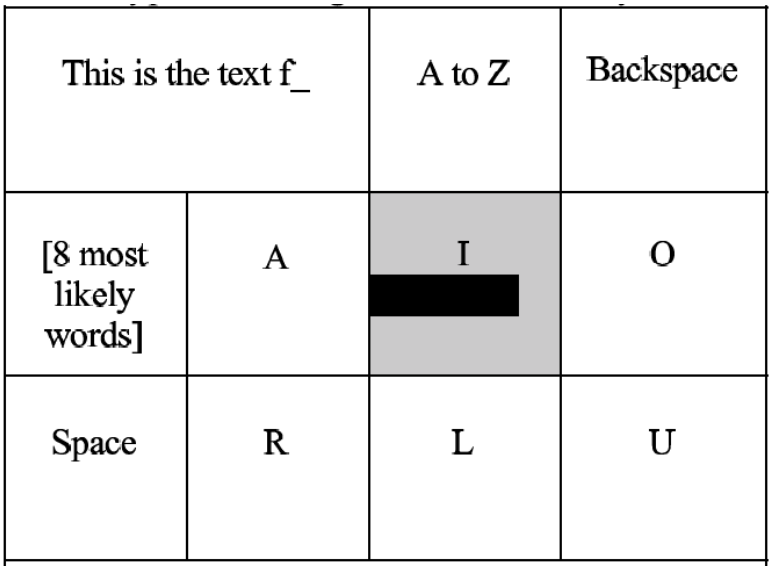


Figure 4.5: GazeTalk's user interface. When the mouse cursor dwells upon a letter for the entire dwell period, it is selected. Here, the mouse is located on letter '`i`', and the dwell period progress bar is almost complete.

lized by individuals with severe disabilities, blinks or winks must be utilized to mean an eye switch. A good example for such an OSK is BlinkWrite [81, 82], which is a scanning ambiguous keyboard that solely makes use of eye blinks for text entry. Fig. 4.3 shows BlinkWrite's user interface. The top row shows the text to be entered by the user. The second row shows part of the text entered by the user so far. The third row shows the letter selection buttons. As seen, the letters are grouped in sizes of 8 or more (except the space key) to minimize the number of items to be scanned. The system makes sure at most two scanning intervals are necessary for text entry. Authors report a text entry throughput of 4.8 wpm with an accuracy of 97% for a scanning interval of 850 ms.

BlinkLink [83] is another example system that uses blinks as input modality in their gaming system and report an accuracy of 96%, which is based on the correct target selection statistics in a coordination game.

OSKs that fall in the second category work by the movements of the

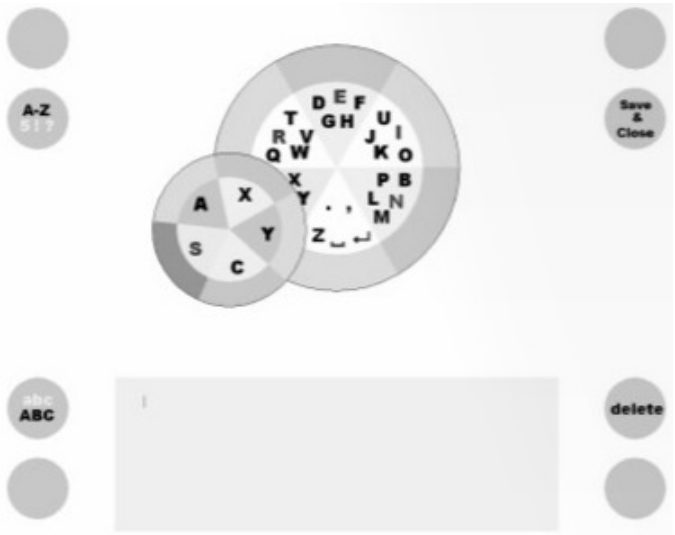


Figure 4.6: pEYEwrite's user interface. The circular pies in the middle are used to enter letters, which appear inside the textbox at the bottom. Other buttons shown on the sides provide auxiliary functionality.

pointing device, that is, the mouse, and are generally called eye typing systems. The idea with these systems is to move the mouse either physically or by an eye tracking software over the item to be selected from the OSK GUI and dwell upon the item for a certain period of time; or alternatively, to move the mouse in certain paths to select items or to draw certain patterns.

One of the earliest examples of such systems is WiVik virtual keyboard [84, 85]. The OSK has a rectangular user interface with a button for each character as shown in Fig. 4.4. To select a button, the user moves the mouse cursor over the button representing the letter and stays over the button for the entire dwell period. The selected character is fed into the system character queue. WiVik also has word prediction to suggest appropriate words to complete the current prefix, which appears on the left side of the user interface. Users may select the desired word from within this suggestion box to make typing faster.

Another similar system is Gazetalk [86, 87]. Unlike WiViK, which displays all letters on the main GUI, Gazetalk uses the statistics of a corpus

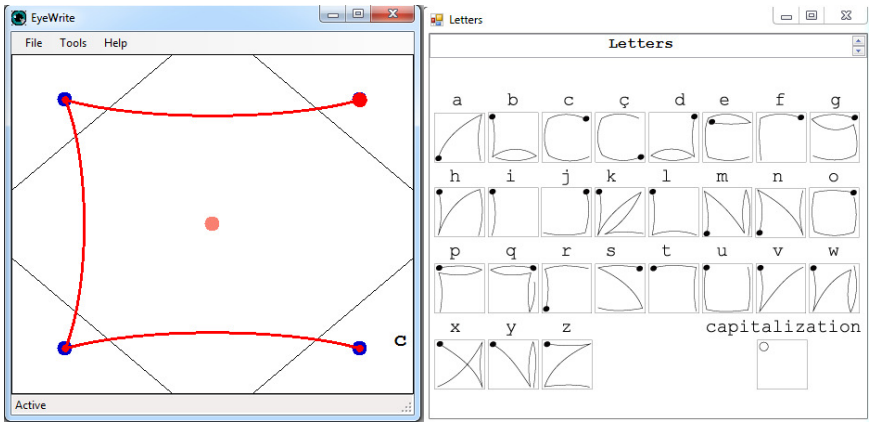


Figure 4.7: EyeWrite's user interface and the patterns representing each character. In the figure, letter 'c' is being typed.

to present the six most widely used letters inside big rectangular boxes on the main GUI as shown in Fig. 4.5. To select a letter, the user moves the cursor over the box representing the letter and stays in that box for the entire dwell period. A progress bar shows the amount of time being waited so far, and the remaining dwell period for the letter to be selected. The transcribed text is shown at the top-left box. Authors report a text entry throughput of 4 wpm with a dwell period of 750 ms.

pEYEwrite [88] has a different design in that it has a circular interface with pie slices as shown in Fig. 4.6. To select a letter, the user first dwells inside the pie containing the desired group of five characters. Once the dwell period is up, a new circular popup menu appears, where each character is displayed inside a separate pie slice. The user dwells inside the pie slice for the desired letter for the second time to select it. The transcribed text appears inside the textbox at the bottom. Other buttons on the user interface provide auxiliary functionality. Authors report a text entry throughput of 7.9 wpm with a dwell period of 400 ms.

EyeWrite [89] is an alternative system, where entering a character involves drawing a certain pattern by moving the mouse rather than dwelling upon an area representing a letter. Patterns resemble hand-printed letters

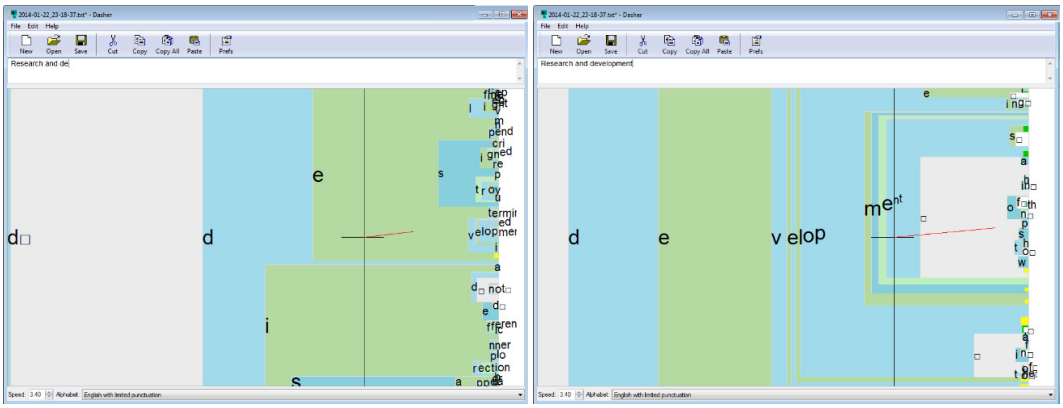


Figure 4.8: Dasher's user interface. Selection is performed by moving the mouse towards a letter. Currently, the word "`development`" is being written.

and are drawn by moving the mouse cursor from one corner of a rectangular window to another as shown in Fig. 4.7. Authors report a text entry throughput of about 5 wpm. In addition to EyeWrite, Minimum Device Independent Text Input Method (MDTIM) [90] and Eye-S [91] also employ this style of "eye graffiti" communication. In these systems, letters are created by a sequence of fixations on several region areas called hotspots, which are usually made invisible and do not interfere with other applications; thus, making it possible to make use of the entire screen area.

Dasher [92] has a radically different design in that it works by continuous gestures. Letters of the alphabet appear on the right hand side of the GUI as shown in Fig. 4.8. Selection is performed by moving the mouse towards a letter, which causes the area representing the letter to get bigger as it is approached, called dynamic zooming. When the letter finally moves to the left of the vertical line dividing the user interface in half, it gets selected. Now, new letters that may follow the selected prefix of letters with respect to a corpus appear on the right. To undo a selection, the mouse must be moved to the left towards the center of the dividing vertical line, which causes the selected letters to move back to right of the vertical line and become deselected. Authors report a text entry throughput of about 20 wpm after one

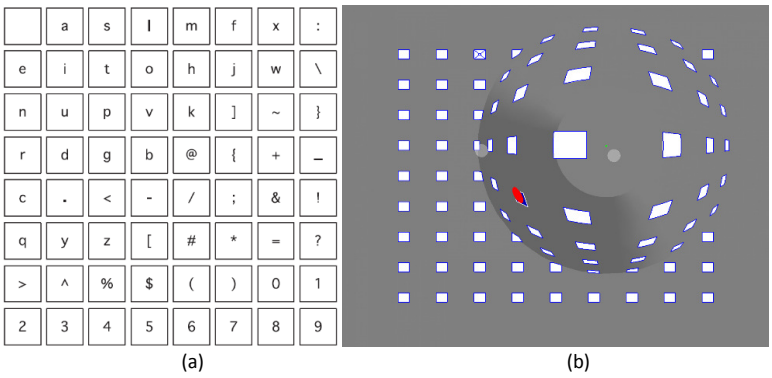


Figure 4.9: (a) A keyboard layout to minimize the number of cursor steps to enter a French book chapter [97], (b) Zooming user interface to make easy and correct selections [99].

hour of practice.

In addition to the OSKs mentioned above, there are many other systems to help disabled persons use computers and enter textual information. For example, [93] presents an OSK with a hierarchical character organization operated by a remote control. VisionKey [94] is one of the earliest OSKs having a somewhat unique selection method. By moving the eye towards the corners of a square grid, the intention is communicated to the OSK. Finally, the selection is performed by moving the mouse to the target cell.

In both scanning keyboards and eye typing systems, reducing the scanning interval or dwelling period will increase the text entry rate, but there will be more errors or missed opportunities for selection. Most systems use a fixed time interval in the range from 500-1500ms. One approach is to dynamically adjust the system's scanning interval or dwelling period based on the previous user performance including text entry throughput, error rate, reaction time etc.

There is also a good body of research on the optimal layout and size of the keys on the OSK user interface [95, 96, 97, 98, 99, 100, 101]. With scanning keyboards, the general idea is to place the most frequently used letters at the beginning of the scan sequence. Authors in [95, 96, 97] evaluate

different letter placement strategies and their effects on the text entry rate. For example, Fig. 4.9(a) shows a keyboard layout optimized to minimize the text entry time for a French book chapter [97]. Zooming interfaces [99, 100] is also an interesting idea in which the region having the current user focus is made bigger for easy and accurate selection. Fig. 4.9(b) illustrates this idea [99]. A study on how auditory and visual feedback affects eye typing is presented in [102]. Authors state that proper feedback by the system facilitates eye typing and influences both the text entry throughput and error rate.

Word or phrase completion or prediction is also widely employed by many OSKs [103, 104, 105, 106, 107, 108, 109]. The idea with word completion is to look at the currently entered word prefix and make word suggestions. The goal is to allow the user enter a complete word without selecting or entering all of its letters, thus increasing the text entry rate. The idea with word prediction or sentence completion is to make use of the words entered so far in the sentence and make better suggestions for the remaining words of the sentence. It is very common to use the last two words, called a bigram, to generate a prediction for the next word, which is also what SliceType does. Authors in [109] look at user interface and human factors affecting the performance of text prediction such as the orientation of the prediction list, number of suggestions to be displayed, proper placing of prediction window etc.

4.3 SliceType: An efficient On-Screen Keyboard

4.3.1 SliceType GUI and Character Layout

In the design of a new OSK, the first design choice is about the shape of the OSK GUI and the placement of the characters on the GUI. There are generally two alternatives for OSK GUI design: (1) Display the characters

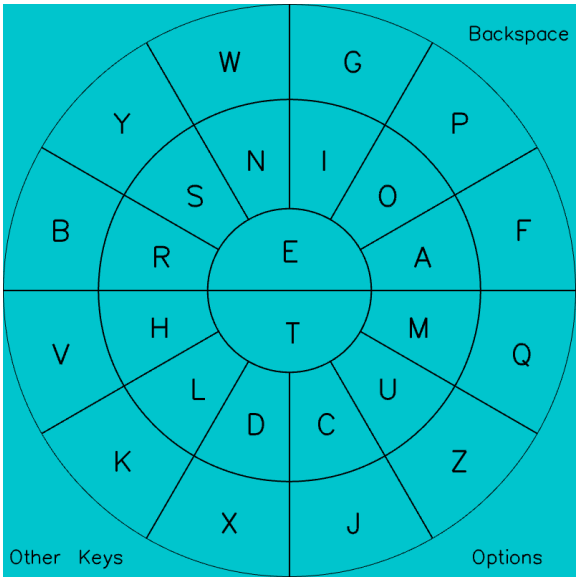


Figure 4.10: SliceType GUI and the layout of the characters on the GUI.

all at once on the GUI, (2) Group the characters under categories, making them reachable by a hierarchical selection mechanism.

Example OSKs in the first category are the Windows scanning keyboard (Fig. 4.1), SoftType (Fig. 4.2), and WiVik (Fig. 4.4). In these OSKs, selecting a character from the GUI requires a single selection event, but the area in which the character is displayed on the GUI is small since there are many characters to display. Example OSKs in the second category are BlinkWrite (Fig. 4.3), GazeTalk (Fig. 4.5), and pEyeWrite (Fig. 4.6). In these OSKs, selecting a character from the GUI requires two selection events: First to select the group of characters containing the desired character, and the second to select the desired character from within that group. The need for an additional selection event for each character slows down typing, but this layout has an advantage: Since only a small number of buttons have to be displayed all at once on the GUI, the area representing each button can be made bigger, which makes it possible to use a coarse input device for selection. On the other hand, displaying all characters on the GUI at the same time results in a smaller footprint for each character assuming that the

overall size of the keyboard is constant. This means that an accurate and high resolution input device is required for accurate selection. Considering that the eye tracking and gaze detection technology has reached some level of maturity and will only improve in the future, we have decided not to use a hierarchical layout for SliceType, but instead chose to display all characters at once on the SliceType GUI as shown in Fig. 4.10.

The keyboard is shaped as a circle encased in a square. The aspect ratio is set to be 1:1, which provides compactness to the keyboard, while minimizing the distance among the characters. The circular keyboard area contains the characters, while each of the remaining four corner areas are used for an additional functionality. The circular area has an innermost circle enclosed by two outer rings. The innermost circle is divided in half to two buttons to store 2 characters, E and T, and the outer rings are divided radially into 24 buttons to store the remaining 24 characters of the English alphabet. The size of each button representing a character has been made to be similar to each other.

Typing on an OSK consists of travelling to and from a set of selection points (buttons) on the keyboard GUI. Some of these points are visited more frequently than others. The main design objective in the placement of the characters on the GUI is to minimize the total distance travelled during typing. An obvious solution to this problem is to assign the more frequently visited points near the center of the GUI. To achieve this objective, we calculated the frequency statistics of the letters in our English corpus, which are given in Fig. 4.11, and used these statistics to lay out the letters on the SliceType GUI as follows: We stored the two most frequently used letters, E and T, in the two halves of the innermost circle. Then the remaining letters have been assigned to the buttons on the two enclosing rings in the order of decreasing frequency starting from inside and moving out. That is, the

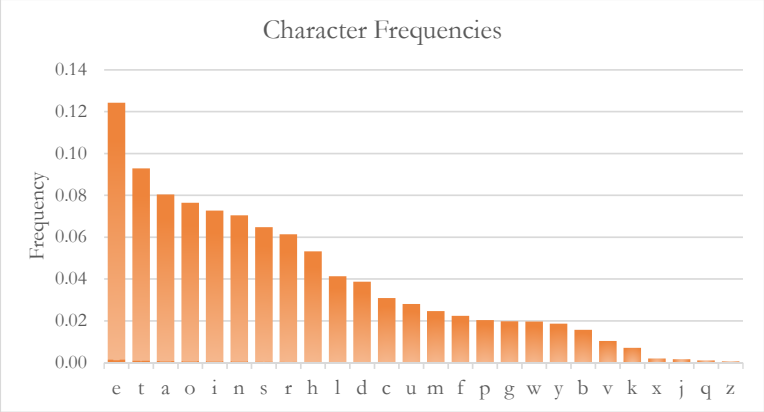


Figure 4.11: Character frequency statistics of the corpus used by SliceType.

most frequently used 12 characters have been stored in the inner ring, and the remaining 12 characters have been stored in the outer ring as shown in Fig. 4.10.

Instead of pairing a more frequently occurring letter with a less frequently occurring letter within the same pie slice as done by SliceType, a better approach may seem like pairing two letters that frequently follow each other in the words of the corpus. Such a choice would obviously result in less mouse movement over the OSK GUI since after a letter is chosen, the next letter would most likely be its pair in the same pie slice. Although this alternative might have easily been implemented in SliceType, we made a conscious decision against this approach due to the following reason: In SliceType, not only do we want to minimize the total amount of mouse movement during typing, which requires frequently occurring letters to be placed close to each other, but we also want to easily and accurately select each letter from the OSK GUI, which requires the screen area representing each letter to be as large as possible. Implementing this second criterion requires displaying as few letters as possible on the OSK GUI so that the screen area representing each letter would be quite large for accurate selection even with a course mouse input mechanism. Recall that this is the goal of a hierarchical OSK GUI such as BlinkWrite, which requires at least two selections for each

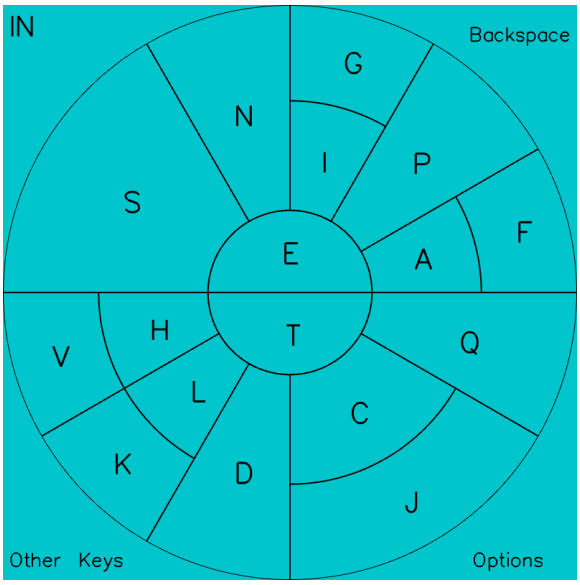


Figure 4.12: Merging of character button spaces based on the current word prefix "`in`". Letter '`w`' has been removed and its button space has been merged with the button space of letter '`n`'.

letter.

To reduce the number of letters displayed on the GUI, SliceType looks at the current word prefix and using its prediction engine, determines possible continuations for the rest of the word. The letters that cannot be used to form a complete word in the corpus from the current prefix are then removed from the GUI. The button space representing a removed letter is then merged with the button space of a neighboring letter so that it grows in size for easy and accurate selection. This behaviour is illustrated in Fig. 4.12: The word prefix entered so far, i.e., "`in`", appears on the upper-left corner of the GUI. Since letter '`y`' cannot follow the prefix "`in`" to make up a legitimate word in the corpus (that is, there is no word that starts with the prefix "`iny`"), letter '`y`' is removed from the SliceType GUI, and its button space is merged with the button space of its neighbor, which is letter '`s`' in this case. Similarly, letter '`w`' is removed and its button space is merged with letter '`n`'. Notice that if letters that frequently follow each other in legitimate words were placed next to each other on the GUI, such button merging would have been less

frequent.

Like many OSKs, SliceType is equipped with word completion capability; that is, SliceType makes use of the currently entered word prefix and provides word suggestions completing the current prefix. To display the suggested words, most OSKs use a separate dedicated area. The user then periodically checks this area to see if the desired word appears in the list, and if it does, s/he selects that word using the OSK selection mechanics. There are some problems with this approach. Firstly, the user must check the word display area frequently to see if the desired word is among the suggested words as they get updated with each new entered letter. This causes many fruitless checks, until the desired word finally appears among the suggestions. This continuous update of the suggested words area may distract novice users, and therefore, they may be inclined to skip checking suggestions altogether either from frustration due to a lack of success in their previous attempts at selecting a suggested word, or to avoid getting distracted because they have to concentrate on selecting the next letter of the word from the GUI. The second problem is about the number of words that needs to be displayed on the suggestions area. For example, if the most likely 10 words are displayed in a list, the user must scan all 10 words for each of their unsuccessful attempt at using the suggestions. Going back and forth between the area displaying the word suggestions and the area displaying the next letter of the word may distract the user and make them mistype a word for an error, which will decrease typing speed although the point of using word completion is to increase typing speed. Furthermore, reserving a separate place on the OSK GUI to display a lot of word suggestions causes waste of valuable GUI space, which can be utilized for some other purpose. If, however, only a few words are suggested and displayed, then the desired word may not appear among the suggested words until after the user types

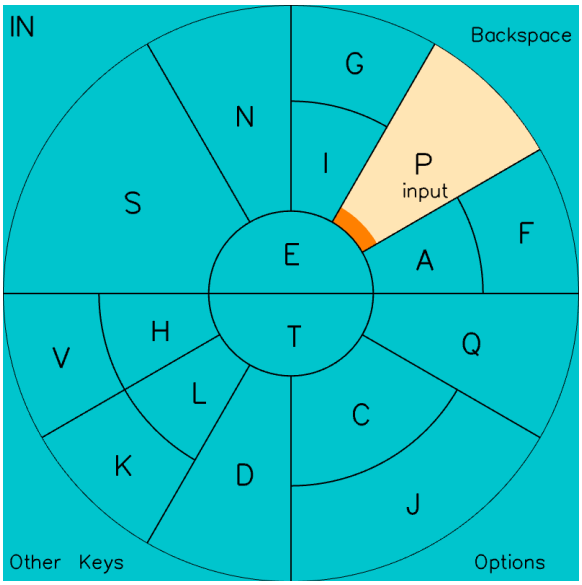


Figure 4.13: Using the current prefix "`in`" and the next letter '`p`', SliceType suggests the word "`input`", which appears inside the pie slice representing letter '`p`'.

the larger portion of the word, which is not preferable either.

Instead of designating a separate GUI area to display word suggestions, we have employed a different approach that enables us to suggest words while not using additional GUI area. Fig. 4.13 illustrates the idea of our novel word suggestion mechanism: As seen on the upper-left corner, the user has so far typed the word prefix "`in`". The user next intends to select letter '`p`' by moving the mouse cursor (either by using a physical mouse or by an eye tracker that translates eye movements to mouse movements) inside the button representing letter '`p`'. Now, not only SliceType highlights the pie slice representing the currently selected letter '`p`' in orange, but it also displays the word "`input`" (generated using the current word prefix "`inp`") suggested by the prediction engine under letter '`p`' within the same pie slice. Displaying the suggested word in the same pie slice as the current letter enables the user to read the suggestion without changing their gaze to a different location, which prevents distractions that may be possible with other OSKs. Notice that SliceType displays only one suggested word within the currently selected

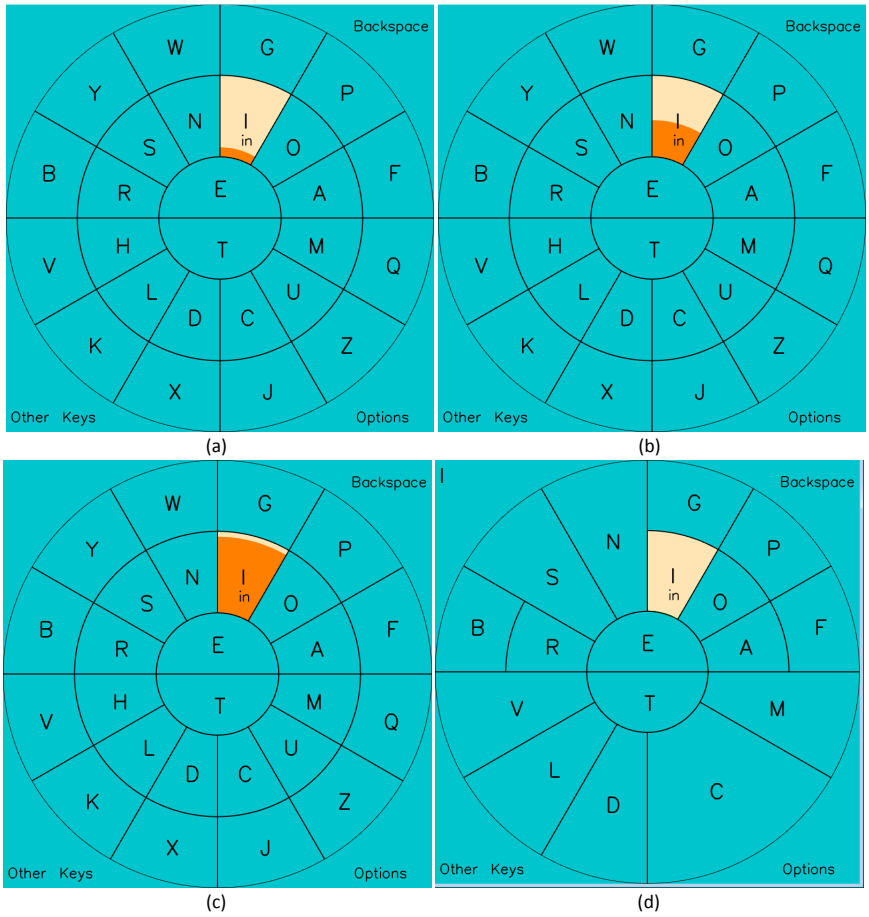


Figure 4.14: (a) The user has just started dwelling upon letter '`i`', which is highlighted with light-orange. The word "`in`" is suggested and displayed inside the button area of letter '`i`'. (b) About 50% of the dwell period is up as indicated by the dark-orange slider, (c) About 90% of the dwell period is up, (d) The entire dwell period is up. Letter '`i`' is selected and appears on the top-left corner of the GUI.

letter. If all suggestions were to be displayed, OSK GUI would be cluttered, which would hinder the user's peripheral vision from locating the next letter. This is why we prefer displaying the word having the highest ranking among the potential suggestions. The details of SliceType's prediction engine is given in section 4.3.3.

4.3.2 Using SliceType

SliceType is a direct selection system, where letter selection is performed by dwelling inside the button representing the letter. The default dwell period is 1000 ms, but it is a parameter that can be adjusted by the user.

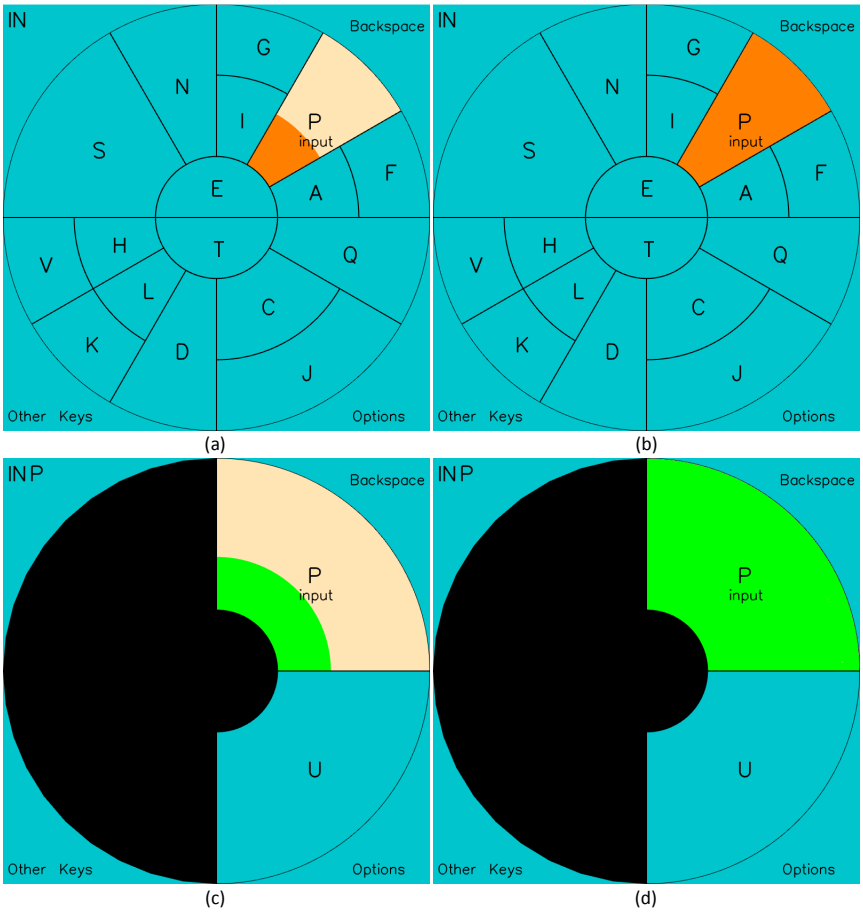


Figure 4.15: The user has entered the word prefix "`in`". (a) The current focus is on letter '`p`' and SliceType suggests the word "`input`" for completion. (b) The user is about to complete the selection of letter '`p`'. (c) The user continues dwelling inside letter '`p`' to select the suggested word. The dwell period progress is illustrated by a green slider instead of an orange slider. (d) The user is about to complete the selection of the suggested word "`input`".

When the mouse cursor is moved inside the boundaries of a button, it is highlighted as its color turns to light-orange. This is illustrated in Fig. 4.14(a), where the user has just moved the mouse inside letter '`i`'. To select this letter, the user has to stay inside the button for the entire dwell period. The amount of dwell period that has passed since the mouse has moved inside the button is indicated by a dark-orange slider that continuously fills the button area. In Fig. 4.14(b), about 50% of the dwell period is up, and in Fig. 4.14(c), about 90% of the dwell period is up as indicated by the dark-orange slider. When the entire dwell period is up, letter '`i`' becomes selected

and appears on the top-left corner of the GUI as shown in Fig. 4.14(d). Also notice in Fig. 4.14(d) that after letter '`i`' is selected, some letters have been removed from the SliceType GUI and their button areas have been merged with the button areas of neighboring letters. The details of this merging algorithm is given in section 4.3.3.

Fig. 4.14 and Fig. 4.15 demonstrate the details of SliceType's word suggestion and selection mechanism. Assume that the user wants to type the word "`input`" using SliceType. As soon as the user moves the cursor inside the first letter of the word, i.e., '`i`', its color changes to light-orange and the first suggested word "`in`" appears inside the button area of letter '`i`'. Since the user desires to type "`input`" rather than "`in`", s/he completes the selection of letter '`i`' by dwelling inside its button area for the entire dwell period, and then moves on to the next letter, i.e., '`n`', of the desired word. Although it is not shown in the figures, SliceType continues suggesting the same word, "`in`"; but this time, the suggested word is displayed inside the button area of letter '`n`'. After letter '`n`' is selected and appears on the upper-left corner of the GUI, the user now moves on to the next letter, i.e., '`p`', of the desired word. This time SliceType suggests the word "`input`", which is displayed inside the button area of letter '`p`' as shown in Fig. 4.15(a). The user stays inside letter '`p`' for the entire dwell period to select it as shown in Fig. 4.15(b), and when the selection is complete, letter '`p`' appears on the upper-left corner of the GUI as shown in Fig. 4.15(c). Now the user wants to select the suggested word "`input`" displayed inside the button area of letter '`p`' so that s/he does not have to type the rest of the letters of the desired word. To select the suggested word, the user must continue dwelling inside letter '`p`' for one more dwell period. This is illustrated in Fig. 4.15(c), where the dwell period progress is illustrated by a green slider to mean the selection of the suggested word instead of an orange slider, which indicates

the selection of the current letter. When this second dwell period is up, the suggested word "`input`" is selected by the user and is sent to the system's character input stream. SliceType GUI then returns back to its default state shown in Fig. 4.10 to display all characters again, and the user can start typing a new word.

SliceType does not have a dedicated area on its GUI to display the user's transcribed text. Instead, it sends the entered text to the system's character input system, which can then be read by the program having the current system focus. This way, the entered text can appear inside any program, e.g., a word processor, a text-to-speech program, a web browser, etc.

Punctuation marks or numeric digits can also be entered using SliceType by selecting "`Other Keys`" functionality located on the lower-left corner of the GUI (refer to Fig. 4.10). SliceType puts a single space character automatically after entered words, unless the last character of the entered text is a punctuation mark or a digit. To manually enter a space character, the user must select the empty area located on the upper-left corner of the GUI. This functionality can also be used to select the current word prefix that appears on the upper-left corner of the GUI to send it to the receiving program. Thus, shorter words that may not appear in the corpus can be typed. For example, if the user wants to type the word "`a`", and the prediction engine suggests the word "`and`" inside the button area of letter '`a`', the user should first select letter '`a`' by dwelling upon its button area, which would then appear on the upper-left corner of the GUI, and then select the button area on the upper-left corner to concur the selection of the word "`a`".

Selecting "`Backspace`", which is located on the upper-right corner of the GUI, has two different connotations: If it is selected by dwelling for a

single dwell period, i.e., when it is filled with a dark-orange slider, it clears the current word prefix displayed on the upper-left corner of the GUI and resets the keyboard to its default state. If it is selected by dwelling for two dwell periods, i.e., when it is filled with a green slider, it deletes the last entered word. To delete the last word from the system's character input stream, SliceType sends the appropriate number of backspace characters. Finally, "`Options`" key located on the lower-right corner of the GUI can be used to adjust dwell period and other parameters of SliceType.

4.3.3 SliceType Operational Mechanics: Button Merging and Word Prediction Engine

We mentioned that if an OSK displays all characters of an alphabet on its GUI at once, then it would take only a single event to select a character; but too many characters on the GUI would also mean that the button area representing each character would be small, and therefore, selecting a button would require high fidelity mouse input. Conversely, displaying only a few characters on the OSK GUI means that the button area representing each character would be large, and therefore, accurate selection of a button can be achieved even with a coarse mouse input; but only a few characters on the GUI would mean multiple events to select each character.

SliceType was designed to combine the benefits of these two design alternatives: Require a single event to select each character, but make the button area representing each character as large as possible for easy and accurate selection. To achieve the first goal, SliceType displays all 26 letters of the English alphabet on its GUI by default. To achieve the second goal, SliceType adaptively decreases the number of letters displayed on its GUI to make the button sizes of the remaining letters larger. Letter removal is achieved by making use of the entered word prefix: After a new letter is

entered by the user, the word prediction engine evaluates the current prefix to make word suggestions. At the same time, the prediction engine determines the letters that cannot follow the current prefix based on its corpus, and those letters are removed from SliceType GUI. One or more of the remaining buttons then take over the areas belonging to the removed buttons. This is called button merging.

During button merging, there are two important considerations: (1) The area belonging to the removed button should be divided fairly among the remaining buttons so that as many buttons as possible can grow in size. One button should not have all the area to itself, as the growth in size yields diminishing returns. (2) As the size of a button grows, the letter displayed on the center of the button should not move too much from its original location. If the button's position changes too much, that would distract the user, and s/he would have a harder time typing as the OSK GUI changes unpredictably. Therefore, the changes to the OSK GUI should be as smooth as possible.

To satisfy both goals above, we designed our button merging algorithm as follows:

(1) Merge radially, i.e., towards the outer ring or the inner ring.

(2) Merge laterally counter-clockwise (ccw) with neighbors on the same ring, only if both have not merged radially.

(3) Merge laterally ccw, only if both have merged radially.

(4) Merge laterally clock-wise (cw), only if they both have merged radially or if they both have not merged radially.

During the button merging, the above algorithm is used in parallel for all buttons. Step (1) is applied for all buttons before proceeding to step (2). This causes preceding steps to be prioritized. In this case, empty spaces will be used for radial merging, rather than lateral merging whenever possible.

The two buttons in the innermost circle simply merge whenever possible, and they do not merge with the rest of the buttons. The rules imply that a button can only merge with its ccw or cw neighbor. This is illustrated in Fig. 4.15(c), where the letters '`p`' and '`u`' do not get merged with the free space on the left side of the OSK GUI. If they were merged with that free space, letters '`p`' and '`u`' would have to be written too far away from their original locations, which would distract the user. Furthermore, that would also achieve little, as the buttons in Fig. 4.15(c) are large enough as they are.

SliceType prediction engine keeps a bigram corpus that consists of word pairs for prediction. The last word entered and the current word prefix are used to predict the word that the user is currently entering. If there are multiple candidates for prediction, the word that belongs to the more frequently used bigram pattern is preferred. Let us illustrate these mechanics with an example. Assume the last word entered by the user is "`his`" and the user brings the cursor over the letter '`f`' to start the next word. The prediction engine looks for the word pairs that start with "`his f-`". There are actually three word pairs that satisfy this condition: "`his first`", "`his father`" and "`his family`". Since the pair "`his first`" is used more frequently compared to the other two, the prediction engine suggests "`first`" for the letter '`f`'.

Note that the word "`for`" is used far more frequently in English language, compared to "`first`". If we had only used a unigram prediction model, the prediction engine would have looked up for the words starting with "`f-`", and would have suggested "`for`" for the letter '`f`'. However, "`his for`" is not a common word pair at all, otherwise the bigram corpus would have included it. This illustrates that one should prioritize bigram prediction results over unigram prediction results. Using a hybrid prediction model (i.e. a corpus that is composed of both single words and word pairs)

would still yield "`for`" as the result, due to the sheer frequency of the word "`for`". However, "`for`" is used so rarely after the word "`his`" (mainly due to grammatical reasons), it should never have the chance to be suggested.

Using n-gram prediction models where n is larger is favorable to achieve more accurate predictions. This is shown by comparing the unigram and bigram predictions above. However, using solely bigram predictions is also not practical. We have mentioned that the bigram corpus has only three members starting with "`his f-`". If the user wanted to write "`his future`", bigram prediction would not have been able to help. For such cases, SliceType uses unigram prediction as a safeguard. Continuing from the previous example, the user has entered "`his`", and wants to enter "`future`" now. When the cursor is on '`f`', prediction engine suggests "`first`", according to the bigram prediction. The user enters '`f`' and moves on to '`u`'. In this case, bigram prediction has no suggestions for "`his fu-`". Hence, unigram prediction takes over. The most frequently used word that starts with "`fu-`" is "`further`", so the unigram prediction suggests it. The user enters '`u`', and moves on to the next letter, '`t`'. The most frequently used word that starts with "`fut-`" is "`future`", and the user can choose the suggestion and finish entering the word. When the corpora are same sized, it is exponentially less likely for corpora with higher n (where n is the size of the word groups) to come up with a result. As even our bigram corpus cannot cover the majority of the common word pairs (like "`his future`"), using corpora with even higher n is not preferred for a project of this scale.

If the corpora would have been kept as a list, finding a prediction would become more complex as the size of the corpora grows. Instead, the unigram and bigram corpora are kept as individual tries, which are constructed to achieve the results as explained above. These tries can be updated by including additional words, or frequencies of the words can be changed based

on usage, which will result in a prediction engine that dynamically adapts.

4.4 Experimental Setup and Results

To compare the performance of SliceType in terms of usability, we have chosen two other publicly available, state of the art OSKs for comparison: GazeTalk [87] and Dasher [92]. These keyboards were chosen to cover a wide variety of design choices. Both SliceType and GazeTalk are operated by the movements of the pointing device and perform dwell-based selection, while Dasher employs continuous gestures approach for operation and selection.

Since all OSKs are operated by the movements of the pointing device, we used an eye tracking system to translate the user's eye movements to mouse movements. The eye tracker used in the experiment is Monocular Edge Analysis System from LC Technologies. It has a sampling rate of 60Hz, and a gaze position accuracy of 0.45 degrees. The users need to calibrate the system before using it, and should not move their head during operation. Experiments were done without a chinrest, as the effect of involuntary movements on calibration was minimal. The calibration was repeated before the use of each keyboard to prevent unfair advantage due to better calibration.

GazeTalk uses the screen area more liberally, and as a result, has fewer and larger buttons. This may be advantageous for participants struggling with eye tracking technology. On the other hand, the other two keyboards require more accurate eye tracking input. Although a keyboard's maximum typing speed may exceed others, constant need of utmost concentration and detection accuracy is a design problem that will negate the speed advantage in daily use. There is certainly a sweet spot for this trade-off, and our experiment was designed to find which keyboard is closer to this sweet spot under daily use conditions.

The typical user of an OSK is expected to use it frequently, and achieve

fair proficiency in a short amount of time. Since the user will be proficient with the keyboard in the majority of the time they use it, it is ideal for the experiment participants to be proficient with each keyboard. However, it is not realistically possible to find participants who are proficient with all the related keyboards. The closest possibility is to teach completely unexperienced novice users. On the other hand, getting a large number of participants to type with each keyboard by gaze for a long enough time to make them proficient is not practical. Instead, we tried to accelerate the participants' learning process by getting them to read about the keyboards beforehand and gave a quick tutorial of the keyboards during one-on-one teaching sessions.

Experiment was performed with 37 undergraduate and graduate computer engineering students. The students were all novice users of the eye tracking system and the keyboards; that is, none of them had used the keyboards nor the eye tracker before. Participants were not asked to remove their glasses or contact lenses. They were given written usage instructions for each keyboard prior to the experiment. After reading the instructions, each participant was individually taught how each keyboard and the eye tracking system work for half an hour. After this brief training session, the participants were asked to eye-type three paragraphs written in intermediate level of English. Each paragraph was chosen to represent daily conversations' level of complexity.

In daily usage, OSKs are usually employed by disabled individuals to communicate their thoughts rather than having them copy a text from a piece of paper to the computer. Therefore, the experimental setup was configured in a way to emulate this real-world usage scenario: Instead of having the participants type a pre-written text, they were read the text aloud as they typed it. Participants were asked to type as much of the text as possible in 5 minutes. Both the order of the usage of keyboards and keyboard-text

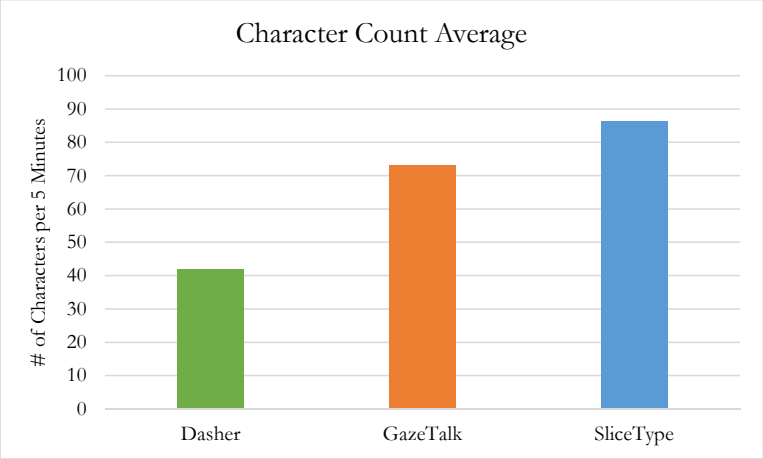


Figure 4.16: Average number of correctly typed characters by the participants in 5 minutes.

matching were permutated to prevent any unfairness. The windows for each keyboard were also kept in a similar size. The parameters for each keyboard were kept in their default values. That is, the dwell period for both SliceType and GazeTalk were set to 1000 ms, and for Dasher the speed was set to 0.8 with adaptive speed adjustment enabled.

After the experiment was finished, the participants were asked which keyboard they would prefer to use if operating an OSK by an eye-tracking software were their only means of verbal communication. The participants put the keyboards in order of preference. The reasoning behind this question was that although a user may type slowly using an OSK, s/he may have a better user experience with that keyboard; that is, a keyboard that provides faster key entry may not always be the most user-friendly one due to human perceptual experience. Both the experiment and the survey were done blindly to prevent any prejudice, bias or inclination towards a particular keyboard, meaning that the participants were not informed about the keyboard developed in this work.

The data obtained from the experiment are the following: (1) the length of the text that a participant was able to type in 5 minutes, (2) the number

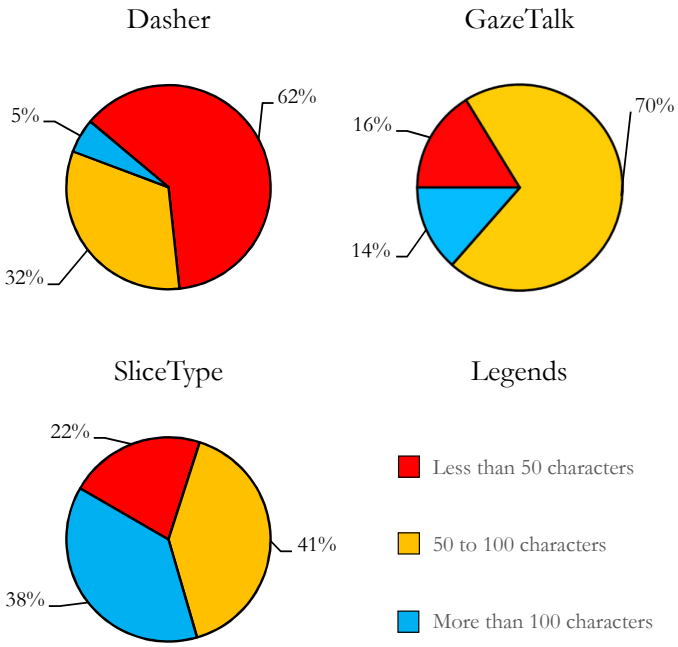


Figure 4.17: Percentage of the participants who typed <50, 50-100, or >100 characters with different keyboards during the 5 minute text entry session.

of errors made during typing, (3) a participant's order of preference of the keyboards in terms of usability. Mistakes in punctuation were also considered as errors, and all consecutive mistakes were counted as a single error. For example, "`How ar-rre you?`" is counted as a single error, even though there are multiple consecutive extra characters. The participants were free to correct their mistakes; only the resulting text at the end of 5 minutes was taken into consideration. It was observed that poor typing performance caused participants to be less likely to correct their mistakes. There were also cases in which the participants did not notice their mistakes, or were unable to correct them even when they tried.

Table 4.1: Text entry rates with each keyboard: (row 1) using a physical mouse for mouse movements, (row 2) using an eye tracker for mouse movements.

Words per minute	Dasher	GazeTalk	SliceType
Previous Experiment [24]	3.50	3.07	5.42
Current Experiment	1.68	2.93	3.45

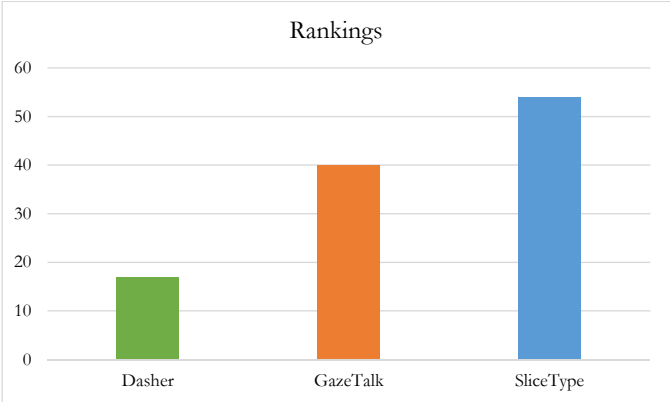


Figure 4.18: Accumulated ratings derived from participants' order of preference for the keyboards.

Fig. 4.16 illustrates the average number of correctly typed characters by all participants in 5 minutes. Clearly, SliceType gives the highest text entry throughput; however, participants' poor performance with Dasher is striking. In our previous experiment [24], Dasher had the second best result. The difference is represented in Table 1, for which the experimental results shown in Fig. 4.16 are converted to words per minute (wpm) using the 5 characters per word standard.

As seen in Table 4.1, SliceType has the highest text entry rate in both experiments, but the ranking between Dasher and GazeTalk is reversed. The difference between the results obtained in this experiment and the results from our previous experiment in [24] has an obvious explanation. The experiment in [24] was conducted using a physical mouse as the input device, while the experiment presented in this paper is conducted with an eye tracking system. Table 4.1 shows that the use of an eye tracker nearly halved the number of words entered using Dasher and SliceType, while it did not

Table 4.2: The average number of errors over all users for each keyboard using an eye tracker.

	Dasher	GazeTalk	SliceType
Average number of errors in 5 minutes	1.03	0.54	0.68

affect GazeTalk's performance much. As mentioned before, both Dasher and SliceType require higher fidelity mouse movement and input, yet GazeTalk's larger buttons allow the use of less accurate input devices. Even though SliceType's performance deteriorated when used with an eye tracking system, it still performs the best among the three keyboards.

In Fig. 4.17, the participants were grouped based on their character entry rate with each keyboard. That is, the percentage of the participants who typed less than 50 characters, in between 50 and 100 characters and more than 100 characters using each keyboard is illustrated. It is clear from the figure that the majority of the users were not able to use Dasher efficiently. This shows the high learning curve of Dasher. With GazeTalk, only a few users failed to type efficiently while most of the users typed at an average typing speed. But only a few users were able to reach very high typing speeds of 100 or more characters. This shows that GazeTalk has a low learning curve and is easy to use. However, with GazeTalk it is not possible to reach very high typing speeds. This must be due to the two-level selection mechanism employed with GazeTalk. With SliceType, many participants were able to reach an average typing speed of 50 to 100 characters, while as many participants were able to reach high typing speeds of more than 100 characters. Only a small percentage of the participants were not able to use SliceType efficiently. This shows that SliceType has a low learning curve, is easy to use, and users can get up to speed with a little practice.

The average number of errors over all users is presented in Table 4.2. As seen from the table, the users made very few errors during typing. This is mainly because the participants tended to correct their mistakes as they noticed them. GazeTalk has the lowest error rate, which has two reasons: (1) in GazeTalk, the entered text takes a large part of the screen, which causes the users to clearly see the text as they entered it. (2) GazeTalk has a slow,

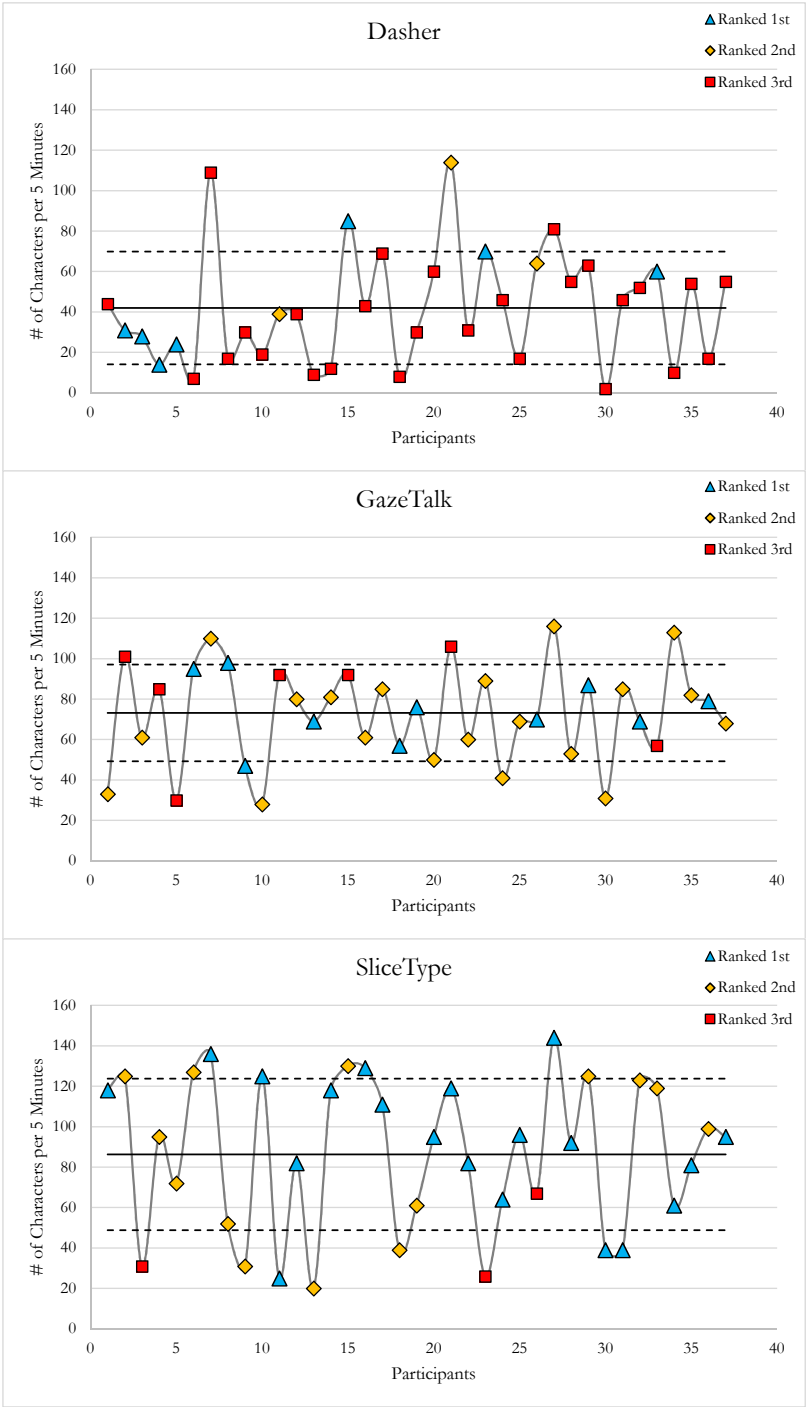


Figure 4.19: Number of characters entered per 5 minutes by each participant, along with the participant's ranking of the keyboard. "Ranked 1st" indicates that the keyboard is the most preferred and "Ranked 3rd" indicates that the keyboard is the least preferred by the participant. The solid black line indicates the mean, while the dashed lines indicate +/-1 standard deviation.

yet simple deletion mechanics. It has a "`Backspace`" button that deletes one character each time it is dwelt upon. On the other hand, SliceType types and deletes entire words instead of single characters. This provides speed, yet causes minor problems in performing deletions. As for Dasher, we have observed that the participants tended not to notice their mistakes during typing, which may be the reason for the high number of errors.

Finally, the participants were asked to rate the keyboards in terms of their order of preference. A keyboard was given 2 points if it was the most preferred keyboard, and 1 point if it was the second most preferred keyboard by each participant. The results are illustrated in Fig. 4.18. Recall that both the experiments and the survey were conducted blindly to prevent any bias or inclination towards any of the keyboards. The participants were also not informed about their relative performance with each keyboard, i.e., the numbers of characters they entered with each keyboard, before filling out the survey. Intuitively, we can expect the users to prefer the keyboards that they can type faster with. The results of the survey can be summarized as follows:

1. 46% of the users ranked the keyboards in the same order with their typing speed.

2. 59% of the participants ranked the keyboard that they typed fastest with as the best.

3. 65% of the participants ranked the keyboard that they typed slowest with as the worst.

While it is obvious that there is a correlation between typing speed and preference, it may not be as strong as it is assumed to be. 41% percent of the users do not prefer the keyboard that they type fastest with. This may indi-

cate that focusing solely on the text entry speed when designing an OSK is rather questionable. 68% of the participants type the fastest with SliceType, while only 54% of the participants prefer SliceType as the best keyboard. This shows that any design improvements to SliceType should be on usability, rather than text entry speed. Participants' individual performances and preference rankings are shown in Fig. 4.19.

4.5 Conclusions

We present an OSK, named SliceType, to help disabled individuals for fast text entry. SliceType has a circular layout with a button area for each character. The selection of a character is performed by moving the mouse inside the button area representing the letter and dwelling on it for the entire dwell period. Mouse movements can be performed by using a physical mouse for people without disabilities, or by an eye tracking system for severely motor impaired individuals for whom the only means of voluntary muscle movements is through their eyes such those suffering from ALS or LIS. SliceType has word completion capability to make typing faster. Experiments performed by a large set of students show that SliceType has the highest user preference and text entry throughput compared to two other state of the art OSKs; namely, GazeTalk and Dasher. For the interested readers, there are video demos and download options for SliceType at our eye tracking webpage: http://ceng.anadolu.edu.tr/cv/eyetracking.

5. CONCLUSIONS

In this thesis study we aim to develop a fully functional eye tracking system from the scratch with all components. We design novel methods for each part and implement them to perform experiments. We present concluding remarks for each study in the following sections.

5.1 Remarks for Proposed Pupil Boundary Detection Algorithm

In the first part, we design and develop a novel pupil boundary detection algorithm to accurately detect pupil center from eye images in real-time. Most of the pupil detection methods in the literature utilize the physical structure of the pupil in a greedy way. Those methods simply threshold the pupil and optimize the remaining pixel cluster by a set of morphological operations. These techniques estimate the pupil center and cannot find the actual pupil border. Therefore they fail in occlusive cases. In the scope of this thesis work we propose a novel pupil detection method which is able to give promising results even in tough occlusive cases.

The algorithm is mainly based on the primitive feature detection and shape recognition techniques. In the first half of the algorithm, we extract elliptical arcs from the image and at the end we search for a consensus of extracted arcs to find the pupil even in severely occluded cases. In the first place, the algorithm tries to find the pupil more quickly by trying to infer that there is no occlusion. To achieve this, it analyses the gradient vector distributions of the segments. If the near-circular segment cannot be found due to an occlusion or view angle, the algorithm extracts all elliptical arcs from the image and searches for a subset that best represents the pupil. In

this way, the algorithm adapts itself and reduce the computation time 60% for the cases which pupil is not occluded.

Our algorithm runs in high speed in eye images where the pupil is clearly visible. If there is any occlusion of eyelids or eyelashes, algorithm detects all elliptical arcs in the image and tries to find a consensus of these arcs for the pupil. This situation constitutes a downside for the algorithm due to the exponential grow rate of pupil candidates for the increasing number of detected elliptical arcs. In such cases, we perform simple heuristics to select among the elliptical arcs to keep the execution time under real-time constraints.

5.2 Remarks for Proposed 3D PoG Estimation Algorithm

For the gaze estimation scheme of the proposed system we develop a novel 3D PoG estimation method based on a head-mounted eye tracking scenario. Our method is easy to deploy in any assistive and mobile application. In the literature remote systems constitutes the major part on the contrary to the head-mounted systems. However, increased interest to the mobile eye tracking, wearable computers and personal displays can increase the demand for head-mounted eye tracking in near future. In this way, head-mounted eye trackers can lead up to hands-free interaction with mobile devices because of the reliability against external factors they provide. Furthermore, the association of gaze with the real world can lead up to many new interaction styles for the user with the environment they live.

Besides the upsides of the proposed approach, there are also a couple of drawbacks. First, our eye tracker suffers from parallax error just like many other head-mounted systems. In most head-mounted systems, there is a scene camera which views the scene from a different point of view than the

user's eye. In other words, there is a phase difference between the eye and scene camera and the visual axes of the user eye and the scene camera are not identical. Therefore, estimated PoG and the actual PoG become dissimilar as the distance between the user and the target changes. In some systems, this problem is solved by optical equipments such as half-transparent mirrors. However, these equipments increase weight of the apparatus, furthermore they cannot be used outdoor due to the vulnerability to the reflections. As a future work we are planning to fix the parallax error with a computational method that aims to compensate the phase difference.

5.3 Remarks for SliceType

In the last part we design and develop a novel on-screen keyboard as a gaze driven application. We can list the contributions of this thesis study as the following:

- The characters which are not available for the current prefix leaves their area to active characters by joining and merging the slices. In this way selection area is extended for the available characters.
- Locations of characters are placed with respect to their frequency in order to maximize the probability of merging (high frequency letter + low frequency letter),
- Merges the slices of characters which are not available in the current context with respect to the current prefix.
- Proposed OSK provides both word completion and word prediction. As a novel contribution, we do not use a separate cell for word completions. Instead, every word candidate is determined with respect to the current prefix and displayed right below each character inside the slice. Thus multiple predictions can be displayed at each step.

- In addition to the word completion with respect to the currently entered prefix, we also provides word prediction with respect to the previously entered words, if available. In this manner word predictions are determined with respect to the previously entered words, not only the current prefix and corpus frequency.

- Proposed OSK design can work in continuous-input by dwelling, therefore can be used without any other mechanism such as extra buttons or switches.

- Keyboard is tested with 37 participants and the results are presented in detail.

SliceType performed the best among the well-known on-screen keyboards in terms of comfort and text throughput. Keeping the letter locations constant and providing the word suggestions right in the letter cells ease the adaptation of the users. Furthermore, wrong letter input ratio caused by the jitter of eye trackers is reduced by merging the cells with respect to the corpus information. In this manner, more frequent letters have larger screen area and the estimated PoG can still fall in the proper cell in spite of the gaze estimation errors.

REFERENCES

[1] Y. Ebisawa, "Improved Video-Based Eye-Gaze Detection Method," IEEE Trans. Instrumentation and Measurement, vol.47, no.4, Aug 1998.

[2] C.H. Morimoto, D. Koons, A. Amir and M. Flickner, "Detection and tracking using multiple light sources," Image and Vision Computing, vol.18, no.4, pp. 331-335, Mar 2000.

[3] J.B. Hiley, A.H. Redekopp, R. Fazel-Rezai, "A Low Cost Human Computer Interface based on Eye Tracking," In Proc. of IEEE EBMS, pg. 3226, 2006.

[4] J.G. Wang, E. Sung, "Study on eye gaze estimation," IEEE Trans. SMC - Part B: Cybernetics, vol.32, no.3, pp.332-350, Jun 2002.

[5] S. Gõni, J. Echeto, A. Villanueva, R. Cabeza, "Robust algorithm for pupil-glint vector detection in a video-occulography eye tracking system," Proc. of Int'l Conf. on Pattern Recognition (ICPR), 2004.

[6] A. Keil, M. Andreas, K. Berger, M.A. Magnor, "Real-Time Gaze Tracking with a Consumer-Grade Video Camera," In Proc. of WSCG, 2010.

[7] J.S. Agustin, H. Skovsgaard, E. Mollenbach, M. Barret, M. Tall, D.W. Hansen, and J.P. Hansen, "Evaluation of a low-cost open-source gaze tracker," In Proc. of ETRA, ACM, 2010.

[8] M. Fischler and R. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," Communications of the ACM, vol. 24, no. 6, pp. 381-395, 1981.

[9] X. Long, O.K. Tonguz, A. Kiderman, "A High Speed Eye Tracking

System with Robust Pupil Center Estimation Algorithm," Proc. of IEEE EMBS, pp.3331-3334, 2007.

[10] L. Lin, L. Pan, L. Wei, and L. Yu, "A Robust and Accurate Detection of Pupil Images," In Proc. of IEEE Biomedical Engineering and Informatics (BMEI), 2010.

[11] L. Ma, T. Tan, Y. Wang, and D. Zhang, "Efficient iris recognition by characterizing key local variations," IEEE Trans. Image Processing, vol.13. no.6, pp.739-750, June 2004.

[12] D. Ballard, "Generalized Hough transform to detect arbitrary patterns," IEEE Trans. Pattern Anal. Machine Intelligence, vol.13, pp. 111-122, 1981.

[13] T. Mäenpää, "An Iterative Algorithm for Fast Iris Detection," Advances in Biometric Person Authentication, LNCS 2005, vol.3781, pp.127-134, 2005.

[14] S. Dey, D. Samanta, "An Efficient Approach for Pupil Detection in Iris Images," Int'l Conf. on Advanced Computing and Communications, pp.382-389, 2007.

[15] D. Zhu, S.T. Moorea, and T. Raphan, "Robust pupil center detection using a curvature algorithm," Computer Methods and Programs in Biomedicine, 59(3), pp.145-157, Elsevier, 1999.

[16] D. Li, D. Winfield, D.J. Parkhurst, "Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches," Proc. of IEEE CVPR - Workshops, 2005.

[17] W.J. Ryan, D.L. Woodard, A.T. Duchowski, and S.T. Birchfield, "Adapting Starburst for Elliptical Iris Segmentation," Proc. of IEEE BTAS, 2008.

[18] N. Kumar, S. Kohlbecher, and E. Schneider, "A Novel approach To Video-Based Pupil Tracking," Proc. of IEEE SMC, 2009.

[19] J. Daugman, "How Iris Recognition Works," Proc. of IEEE ICIP, vol.1, pp. 33-36, 2002.

[20] E.M. Arvacheh, H.R. Tizhoosh, "IRIS Segmentation: Detecting Pupil, Limbus and Eyelids," Proc. of IEEE ICIP, pp.2453-2456, 2006.

[21] Y. Ebisawa, "Robust pupil detection by image difference with positional compensation," In Proc. of IEEE VECIMS, pp.143-148, 2009.

[22] D.W. Hansen, Q. Ji, "In the Eye of the Beholder: A Survey of Models for Eyes and Gaze," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 32, no. 3, pp.478-500, Mar. 2010.

[23] C.H. Morimoto, M.R.M. Mimica, "Eye gaze tracking techniques for interactive applications," Computer Vision and Image Understanding, vol.98, pp. 4-24, 2005.

[24] C. Topal and C. Akinlar, "Edge Drawing: A Combined Real-Time Edge and Segment Detector," Journal of Visual Communication and Image Representation, 23(6), 862-872, 2012.

[25] C. Topal, C. Akinlar, Y. Genc, Edge Drawing: A Heuristic Approach To Robust Real-time Edge Detection, The Twentieth Int'l Conf. Pattern Recognition (ICPR), pp. 2424-2427, 2010.

[26] C. Topal, O. Ozsen, C. Akinlar, Real-time edge segment detection with edge drawing algorithm, Int'l Symp. on Image and Signal Processing and Analysis (ISPA), Dubrovnik, Croatia, September 2011.

[27] Edge Drawing website and online demo page. http://ceng.anadolu.edu.tr/CV/EdgeDrawing. Accessed on 27.Agu.2013.

[28] A. Desolneux, L. Moisan, J.M. Morel, Edge detection by Helmholtz principle, Journal of Mathematical Imaging and Vision, 14(3), 271-284, 2001.

[29] A. Desolneux, L. Moisan, J.M. Morel, Meaningful Alignments, International Journal of Computer Vision, 40(1), pp.7-23, 2007.

[30] C. Akinlar, C. Topal, EDPF: A Real-time Parameter-free Edge Segment Detector with a False Detection Control, International Journal of Pattern Recognition and Artificial Intelligence, 26 (1), 2012.

[31] EDPF website and online demo page.
http://ceng.anadolu.edu.tr/CV/EDPF. Accessed on 27.Agu.2013.

[32] L. Jia and L. Kitchen, "Object-Based Image Similarity Computation Using Inductive Learning of Contour-Segment Relations," Trans. Image Processing, 9(1), pp.80-87, Jan. 2000.

[33] M. Shia, Y. Fujisawab, T. Wakabayashia, F. Kimura, "Handwritten numeral recognition using gradient and curvature of gray scale image," Pattern Recognition 35(10), pp.2051-2059, Oct. 2002.

[34] A.C. Jalba, M.H. F. Wilkinson, J.B.T.M. Roerdink, "Shape Representation and Recognition Through Morphological Curvature Scale Spaces," Trans. Image Processing, 15(2), pp.331-341, Feb. 2006.

[35] V. Ferrari, L. Fevrier, F. Jurie, and C. Schmid, "Groups of Adjacent Contour Segments for Object Detection," IEEE Trans. PAMI, 30(1), Jan. 2008.

[36] C. Martinez-Ortiz, J. Žunić, "Curvature weighted gradient based shape orientation ," Pattern Recognition, 43(9), pp.3035-3041, Sep. 2010.

[37] A. Fitzgibbon, M. Pilu, and R.B. Fisher, "Direct Least Square Fitting of Ellipses," IEEE Trans. Pattern Anal. Machine Intelligence, vol. 21,

no. 5, May 1999.

[38] R. Nürnberg, Distance from a Point to an Ellipse, 2006. Online available at http://www2.imperial.ac.uk/ rn/distance2ellipse.pdf. Accessed on 27.Agu.2013.

[39] S. Ramanujan, "Collected Papers of Srinivasa Ramanujan," Chelsea Publishing, New York, 1962.

[40] C. Akinlar, C. Topal, "EDCircles: A real-time circle detector with a false detection control," Pattern Recognition, 46(3), 725-740, March 2013.

[41] EDCircles website and online demo page. http://ceng.anadolu.edu.tr/CV/EDCircles. Accessed on 27.Agu.2013.

[42] C. Topal, K. Özkan, B. Benligiray, C. Akinlar, "A Robust CSS Corner Detector based on the Turning Angle Curvature of Image Gradients," IEEE Int'l Conf. Acoustics, Speech and Signal Processing (ICASSP), Vancouver, CANADA, 2013.

[43] Anadolu CVLab Eye tracking research website. http://ceng.anadolu.edu.tr/CV/EyeTracking. Accessed on 27.Agu.2013.

[44] C. Topal, B. Benligiray, C. Akinlar, "On The Efficiency Issues of Virtual Keyboard Design," In Proc. of IEEE Int'l Conf. VECIMS, Tianjin, CHINA, 2012.

[45] L. R. Young and D. Sheena, "Methods & Designs Survey of eye movement recording methods," Behavior Research Methods & Instrumentation, vol. 7(5), pp.397 - 429, 1975.

[46] A. T. Duchowski, "A breadth-first survey of eye tracking applications, Behav. Res. Methods Instrum. Comput., vol. 34, pp. 1-16, 2002.

[47] K.P. White Jr., T.E. Hutchinson, and J.M. Carley, "Spatially Dynamic Calibration of an Eye-Tracking System," IEEE Trans. Systems, Man, and Cybernetics, vol. 23, no. 4, 1993.

[48] D.W. Hansen, J.P. Hansen, M. Nielsen, A.S. Johansen, and M.B. Stegmann, "Eye Typing Using Markov and Active Appearance Models," Proc. IEEE Workshop Applications on Computer Vision, pp. 132-136, 2003.

[49] Q. Ji and Z. Zhu, "Eye and Gaze Tracking for Interactive Graphic Display," Proc. Second Int'l Symp. Smart Graphics, pp. 79-85, 2002.

[50] Z. Zhu and Q. Ji, "Novel Eye Gaze Tracking Techniques under Natural Head Movement," IEEE Trans. Biomedical Engineering, vol. 54, no. 12, pp. 2246-2260, Dec. 2007.

[51] C.H. Morimoto, D. Koons, A. Amir, and M. Flickner, "Pupil Detection and Tracking Using Multiple Light Sources," Image and Vision Computing, vol. 18, no. 4, pp. 331-335, 2000.

[52] J. Sigut and S. Sidha, "Iris Center Corneal Reflection Method for Gaze Tracking Using Visible Light," IEEE Trans. Biomedical Engineering, vol. 58, no. 2, February 2011.

[53] C. Topal, S. Gunal, O. Kocdeviren, A. Dogan, and O.N. Gerek, "A Low-Computational Approach on Gaze Estimation With Eye Touch System," IEEE Trans. Cybernetics, vol. 44, no. 2, February 2014.

[54] D. Beymer and M. Flickner, "Eye Gaze Tracking Using an Active Stereo Head," Proc. IEEE Conf. Computer Vision and Pattern Recognition, vol. II, pp. 451-458, 2003.

[55] T. Ohno, N. Mukawa, and A. Yoshikawa, "Freegaze: A Gaze Tracking System for Everyday Gaze Interaction," Proc. Eye Tracking Research

Applications Symp., pp. 125-132, 2002.

[56] K. Talmi and J. Liu, "Eye and Gaze Tracking for Visually Controlled Interactive Stereoscopic Displays," Signal Processing: Image Comm., vol. 14, no. 10, pp. 799-810, 1999.

[57] S.-W. Shih, Y.-T. Wu, and J. Liu, "A Calibration-Free Gaze Tracking Technique," Proc. 15th Int'l Conf. Pattern Recognition, pp. 201-204, 2000.

[58] S.-W. Shih, J. Liu, "A Novel Approach to 3-D Gaze Tracking Using Stereo Cameras," IEEE Trans. Systems, Man, and Cybernetics, vol. 34, no. 1, pp. 234-245, Feb. 2004.

[59] B. Noureddin, P.D. Lawrence, and C.F. Man, "A Non-Contact Device for Tracking Gaze in a Human Computer Interface," Computer Vision and Image Understanding, vol. 98, no. 1, pp. 52-82, 2005.

[60] E.D. Guestrin and M. Eizenman, "General Theory of Remote Gaze Estimation Using the Pupil Center and Corneal Reflections," IEEE Trans. Biomedical Eng., vol. 53, no. 6, pp. 1124-1133, June 2006.

[61] A. Villanueva and R. Cabeza, "A Novel Gaze Estimation System With One Calibration Point," IEEE Trans. Systems, Man, and Cybernetics - Part:B Cybernetics, vol. 38, no. 4, Aug. 2008.

[62] A. Villanueva and R. Cabeza, "Models for Gaze Tracking Systems," J. Image and Video Processing, vol. 2007, no. 3, 2007.

[63] L.H. Yu and M. Eizenman, "A New Methodology for Determining Point-of-Gaze in Head-Mounted Eye Tracking Systems," IEEE Trans. Biomedical Engineering, no. 10, vol. 51, October.

[64] R. Valenti, N. Sebe, and T. Gevers, "Combining Head Pose and Eye Location Information for Gaze Estimation," IEEE Trans. Image Pro-

cessing, vol. 21, no. 2, February 2012.

[65] C. Topal and C. Akinlar, "An Adaptive Algorithm for Precise Pupil Boundary Detection Using the Entropy of Contour Gradients," submitted for publication. Available online: ceng.anadolu.edu.tr/cv/eyetracking/download/PupilDetection.pdf.

[66] B. Benligiray, C. Topal, C. Akinlar, "EDMarkers: A Scalable and Robust Fiducial Marker System for Augmented Reality Applications," submitted for publication.

[67] D. W. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," SIAM Journal of Applied Mathematics, vol.11, no.2, 1963.

[68] G. Bradski and A. Kaehler, "Learning OpenCV Computer Vision with the OpenCV Library," O'Reilly Media, September 2008.

[69] R. Hartley and A. Zisserman, "Multiple View Geometry in Computer Vision," Second Ed., Cambridge Univ. Press, 2004.

[70] Eye tracking research web page: http://ceng.anadolu.edu.tr/cv/eyetracking. Accessed on 15^{th} July 2014.

[71] J. Velez, J.D. Borah, "Visor and Camera Providing A Parallax-Free Field-Of-View Image For A Head-Mounted Eye Movement Measurement System", U.S. Patent 4 852 988, Aug. 1, 1989.

[72] D. Li, "Low-cost eye-tracking for human computer interaction", M.Sc. Thesis, Iowa State University, Ames, IA, 2006.

[73] D. Mardanbegi and D.W. Hansen, "Real-time parallax error compensation in head-mounted eye trackers", The Scandinavian Workshop on Applied Eye Tracking (SWAET), 2012.

[74] K.M. Evans, R.A. Jacobs, J.A. Tarduno, J.B. Pelz, "Collecting and Analyzing Mobile Eye-tracking Data in Outdoor Environments," Journal of Eye Movement Research, vol.5, iss.2, 2012.

[75] J.K Jacob, "The Uses of Eye Movements in Human-Computer Interaction Techniques: What You Look At is What You Get," ACM Transactions on Information Systems, vol. 9, no. 3, pp. 152-169, 1991.

[76] P. Majaranta and K-J. Raiha, "Twenty Years of Eye Typing: Systems and Design Issues," in Proc. Eye Tracking Research and Applications (ETRA), New Orleans LA, pp. 15-22, 2002.

[77] A. Mukherjee, S. Bhattacharya, P. Haider, and A. Basu, "A virtual predictive keyboard as a learning aid for people with neuro-motor disorders," Advanced Learning Technologies, pp. 1032- 1036, 2005.

[78] S. Sarcar, S. Ghosh, P.K. Saha and D. Samanta, "Virtual keyboard design: State of the arts and research issues," in Proc. IEEE Students' Technology Symposium (TechSym), pp. 289-299, 2010.

[79] V. Prabhu, and G. Prasad, "Designing a virtual keyboard with multimodal access for people with disabilities," Information and Communication Technologies (WICT), pp. 1133-1138, Dec. 2011.

[80] P.E. Jones, "Virtual keyboard with scanning and augmented by prediction," In. Proc. Of the 2nd European Conference on Disability, Virtual Reality and Associated Technologies, University of Reading, U.K., pp. 45-51, 1998.

[81] B. Ashtiani and I.S. MacKenzie, "BlinkWrite2: an improved text entry method using eye blinks," in Proc. Eye Tracking Research and Applications (ETRA), pp. 339-345, 2010.

[82] I.S. MacKenzie and B. Ashtiani, "BlinkWrite: efficient text entry using

eye blinks," Inf. Soc., vol. 10, pp. 69-80, 2011.

[83] K. Grauman, M. Betke, J. Lombardi, J. Gips and G.R. Bradski, "Communication via eye blinks and eyebrow raises: video-based human computer interfaces," Univ Access Inform Soc., vol. 2, pp. 359-373, 2003.

[84] F. Shein, G. Hamann, N. Brownlow, J. Treviranus, D. Milner and P. Parnes, "WiVik: A visual keyboard for Windows 3.0," In Proc. the Annual Conference of the Rehabilitation Engineering Society of North America (RESNA), Arlington, VA, pp. 160162, 1991.

[85] http://www.wivik.com/.

[86] J.P. Hansen, D.W. Hansen, A.S. Johansen, "Bringing gaze based interaction back to basics," in Proc. HCI International, Erlbaum, Mahwah, NJ, pp. 325328, 2001.

[87] J.P. Hansen, A.S. Johansen, D.W. Hansen, K. Itoh and S. Mashino, "Language technology in a predictive, restricted on-screen keyboard with ambiguous layout for severely disabled people," in Proc. EACL Workshop on Language Modeling for Text Entry Methods, pp. 59-66, 2003.

[88] A. Huckauf and M.H. Urbina, "Gazing with pEYEs: towards a universal input for various applications," in Proc. Eye Tracking Research and Applications (ETRA), pp.51-54, 2008.

[89] J.O. Wobbrock, J. Rubinstein, M.W. Sawyer and A.T. Duchowski, "Longitudinal evaluation of discrete consecutive gaze gestures for text entry," in Proc. Eye Tracking Research and Applications (ETRA), pp. 1119, 2008.

[90] P. Isokoski and R. Raisamo, "Device independent text input: a rationale and an example," in Proc. The Working Conference on Advanced

Visual Interfaces (AVI), New York, pp. 7683, 2000.

[91] M. Porta, M. Turina and S. Eye, "A full-screen input modality for pure eye-based communication," in Proc. Eye Tracking Research and Applications (ETRA), New York, pp. 2734, 2008.

[92] D. J. Ward, A.F. Blackwell and D.J.C. MacKay, "Dasher: a data entry interface using continuous gestures and language Models," in Proc. the ACM Symposium on User Interface Software and Technology (UIST), New York, pp. 129137, 2000.

[93] D. Kar, A. Pal, C. Bhaumik, J. Shukla and S. Ghoshdastidar, "A Novel On-Screen Keyboard for Hierarchical Navigation with Reduced Number of Key Strokes," in Proc. IEEE International Conference on Systems, Man. and Cybernetics (SMC), pp. 5072-5076, 2009.

[94] D.A. Kahn, J. Heynen and G.L. Snuggs, "Eye-Controlled Computing: The VisionKey Experience," in Proc. The Fourteenth International Conference on Technology and Persons with Disabilities (CSUN), 1999.

[95] H.S. Venkatagiri, "Efficient keyboard layouts for sequential access in augmentative and alternative communication," Augment. Altern. Communication, vol. 15, pp. 126134, 1999.

[96] S. Zhai, M. Hunter and B.A. Smith, "Performance Optimization of Virtual Keyboards," HumanComputer Interaction, vol. 17, no. 2-3, pp. 229-269, 2002.

[97] G. Francis, and E. Johnson, "Speed - accuracy tradeoffs in specialized keyboards," Int. J. Human-Computer Studies, vol. 69, pp. 526-538, 2011.

[98] S. Jain and S. Bhattacharya, "Virtual keyboard layout optimization," in Proc. IEEE Students' Technology Symposium (TechSym), pp. 312-

317, 2010.

[99] M. Ashmore, A.T. Duchowski, and G. Shoemaker, "Efficient eye pointing with a fisheye lens," In Proc. Graphics Interface, Canadian Information Processing Society, Toronto, pp. 203-210, 2005.

[100] R. Bates and H. Istance, "Zooming Interfaces!: Enhancing the Performance of Eye Controlled Pointing Devices," in Proc. Assistive Technologies, pp. 119126, 2002.

[101] S. Ghosh, S. Sarcar, M.K. Sharma and D. Samanta, "Effective virtual keyboard design with size and space adaptation," in Proc. IEEE Students' Technology Symposium (TechSym), pp. 262-267, 2010.

[102] P. Majaranta, I.S. MacKenzie, A. Aula and K. Räihä, "Auditory and visual feedback during eye typing," in Proc. CHI Extended Abstracts, pp. 766-767, 2003.

[103] G. Foster, P. Langlais and G. Lapalme, "User-friendly text prediction for translators," in Proc. the Conference on Empirical Methods in Natural Language Processing, 2002.

[104] N. Jacobs and H. Blockeel, "Sequence prediction with mixed order Markov chains," in Proc. the Belgian/Dutch Conference on Artificial Intelligence, 2003.

[105] N. Garay-Vitoria and J. Abascal, "A comparison of prediction techniques to enhance the communication of people with disabilities," in Proc. the 8th ERCIM Workshop User Interfaces For All, 2004.

[106] K. Grabski and T. Scheffer, "Sentence completion," in Proc. the ACM SIGIR Conference on Information Retrieval, 2004.

[107] S. Bickel, P. Haider and T. Scheffer, "Predicting Sentences using N-Gram Language Models," in Proc. the Conference on Empirical Meth-

ods in Natural Language Processing (EMNLP), 2005.

[108] I.S. MacKenzie and X. Zhang, "Eye typing using word and letter prediction and a fixation algorithm", in Proc. Eye Tracking Research and Applications (ETRA), pp. 55-58, 2008.

[109] M.K. Sharma, S. Dey, P.K Saha and D. Samanta, "Parameters effecting the predictive virtual keyboard," in Proc. IEEE Students' Technology Symposium (TechSym), pp. 268-275, 2010.