A Gaze Estimation Method and System for Natural Light Cameras

Onur Ferhat

Universitat Autònoma
de Barcelona

A Gaze Estimation Method and System for Natural Light Cameras

Director

Dr. Fernando Vilariño
Dept. Ciències de la Computació & Centre de Visió per Computador
Universitat Autònoma de Barcelona

A dissertation submitted by **Onur Ferhat** at Dept. Ciències de la Computació, Universitat Autònoma de Barcelona to fulfil the degree of **Doctor of Philosophy** within the Computer Science Doctorate Program.

Bellaterra, June 13, 2017

There's no map
And a compass wouldn't help at all
— Björk

It is good to have an end to journey toward;
but it is the journey that matters, in the end.
— Ursula K. Le Guin

To my family, friends and Zeynep...

Acknowledgements

Six years ago, I set off for the journey of my life: quitting my job, leaving all my colleagues and friends behind, to pursue a dream that I held on to. During all these long years, one may even have a hard time remembering what he was searching for in the first place. Now, when I'm about to finish this chapter of my life, it all becomes clear: it's not about reaching somewhere, the journey is what matters. Here I want to take my time to remember all those that were with me during these years, who helped me carry on, keep learning and investigating, and of course, enjoy life to the fullest.

First of all, I would like to thank my supervisor, Fernando Vilariño, who gave me this opportunity in the first place. His patience, support and trust in me has been key in the completion of this thesis today.

Having a great work environment provided me with the peace of mind that I needed to concentrate on my work. For this, I would first like to thank the staff and colleagues at Computer Vision Center, where I spent my first five years. Since October 2016, Criteo Barcelona office has been my second home, and I'm grateful for all the comfort, peace and joy that working there with my great colleagues entails.

Special thanks go to Gianluca Daga & Fabrizio Cristiano, who have been very generous by helping me get my new computer, which was much needed for the last push I had to make in this work. Also appreciated is Reme's help, who appeared just when I needed it the most.

My time working with Arcadi Llanza and Dan Norton has taught me many things. I would like to thank them for their collaboration, as it stirred my imagination for new ideas. I'm also especially grateful for all the great friends I had throughout all these years, from Shell & Adela in my first year, to Yaxing, Lichao, Xialei, Lu, Dena, Hana and Felipe more recently. Outside CVC, İnci, Alican, Civan, and Ceren, with whom I shared many great moments in lunches and dinners, made my experience even more pleasant.

Coming to the closer circle of friends, I would like to thank Aziz, Erdal, Hüma & Sima sisters, Gözde, Aslı & Jonatan and Zeynep Ergönenç, who have made this a unique experience. Without them, Barcelona would be just a nice place from a touristic perspective, and their presence turned it into a home for me. Thanks for making this possible and enabling me to go further in my journey. Bojana, Arash, Francesco and Gemma have been amazing companions during the last years; facing the same challenges (and surviving in the same basement!) bound us together, and made it possible to keep going further. And my gratitude to Germán, who was there

from the beginning till the end, and who has been like a pillar supporting the sky.

To finish my acknowledgements, I want to thank my family, who have always been patient and supportive during the last years. And lastly, to Zeynep, who stepped into my life at the last stage of this path, for giving me all the love, energy and purpose I needed to finish this work.

Abstract

Eye tracker devices have traditionally been only used inside laboratories, requiring trained professionals and elaborate setup mechanisms. However, in the recent years the scientific work on easier–to–use eye trackers which require no special hardware—other than the omnipresent front facing cameras in computers, tablets, and mobiles—is aiming at making this technology common–place. These types of trackers have several extra challenges that make the problem harder, such as low resolution images provided by a regular webcam, the changing ambient lighting conditions, personal appearance differences, changes in head pose, and so on. Recent reasearch in the field has focused on all these challenges in order to provide better gaze estimation performances in a real world setup.

In this work, we aim at tackling the gaze tracking problem in a single camera setup. We first analyze all the previous work in the field, identifying the strengths and weaknesses of each tried idea. We start our work on the gaze tracker with an appearance–based gaze estimation method, which is the simplest idea that creates a direct mapping between a rectangular image patch extracted around the eye in a camera image, and the gaze point (or gaze direction). Here, we do an extensive analysis of the factors that affect the performance of this tracker in several experimental setups, in order to address these problems in future works. In the second part of our work, we propose a feature–based gaze estimation method, which encodes the eye region image into a compact representation. We argue that this type of representation is better suited to dealing with head pose and lighting condition changes, as it both reduces the dimensionality of the input (i.e. eye image) and breaks the direct connection between image pixel intensities and the gaze estimation. Lastly, we use a face alignment algorithm to have robust face pose estimation, using a 3D model customized to the subject using the tracker. We combine this with a convolutional neural network trained on a large dataset of images to build a face pose invariant gaze tracker.

Key words: *eye tracking, gaze tracking, human computer interaction, computer vision*

Resumen

Los dispositivos de seguimiento de ojos han sido tradicionalmente utilizados sólo dentro de los laboratorios, y requerían profesionales capacitados y mecanismos de configuración elaborados. Sin embargo, en los últimos años el trabajo científico sobre dispositivos fáciles de usar que no requieren ningún hardware especial—aparte de las omnipresentes cámaras frontales en ordenadores, tabletas y móviles—tiene como objetivo hacer esta tecnología accesible. Estos tipos de seguidores tienen varios desafíos adicionales que dificultan el problema, como imágenes de baja resolución proporcionadas por una cámara web normal, las condiciones cambiantes de iluminación ambiental, las diferencias de apariencia personal, los cambios en la postura de la cabeza, etc. La investigación reciente en el campo se ha centrado en todos estos desafíos con el fin de proporcionar mejores resultados de estimación de la mirada en una configuración del mundo real.

En este trabajo, tratamos de abordar el problema de seguimiento de la mirada en una configuración de cámara única. Primero analizamos todo el trabajo previo en el campo, identificando las fortalezas y debilidades de cada idea probada. Comenzamos nuestro trabajo con un método de estimación de la mirada basado en la apariencia, que es la idea más simple que crea una correlación directa entre un parche de imagen rectangular extraído alrededor del ojo en una imagen de cámara y el punto de mirada. Aquí, hacemos un extenso análisis de los factores que afectan el desempeño de este seguidor en varias configuraciones experimentales, con el fin de abordar estos problemas en futuros trabajos. En la segunda parte de nuestro trabajo, proponemos un método de estimación de la mirada basado en características, que codifica la imagen de la región ocular en una representación compacta. Argumentamos que este tipo de representación es más adecuado para tratar con la pose de la cabeza y cambios en la condición de iluminación, ya que reduce la dimensionalidad de la entrada (es decir, la imagen del ojo) y rompe la conexión directa entre las intensidades de los píxeles de la imagen y la estimación de la mirada. Por último, utilizamos un algoritmo de alineación de la cara para tener una estimación de la postura de cara robusta, usando un modelo 3D personalizado para el sujeto que usa el seguidor. Combinamos esto con una red neuronal convolucional entrenada en un gran conjunto de datos de imágenes para conseguir un seguidor de miradas invariante a la postura de la cara.

Palabras clave: *seguimiento de los ojos, seguimiento de la mirada, interacción de la computadora humana, visión por computador*

Resum

Els dispositius de seguiment d'ulls han estat tradicionalment utilitzats només dins dels laboratoris, i requerien professionals capacitats i mecanismes de configuració elaborats. No obstant això, en els últims anys el treball científic sobre dispositius fàcils d'usar que no requereixen cap maquinari especial—a part de les omnipresents càmeres frontals en ordinadors, tauletes i mòbils —té com a objectiu fer aquesta tecnologia accessible. Aquests tipus de seguidors tenen diversos desafiaments addicionals que dificulten el problema, com imatges de baixa resolució proporcionades per una càmera web normal, les condicions canviants d'il·luminació ambiental, les diferències d'aparença personal, els canvis en la postura del cap, etc. La investigació recent en el camp s'ha centrat en tots aquests desafiaments amb la finalitat de proporcionar millors resultats d'estimació de la mirada en una configuració del món real.

En aquest treball, tractem d'abordar el problema de seguiment de la mirada en una configuració de càmera única. Primer analitzem tota la feina prèvia al camp, identificant les fortaleses i debilitats de cada idea provada. Comencem el nostre treball amb un mètode d'estimació de la mirada basat en l'aparença, que és la idea més simple que crea una correlació directa entre un pegat d'imatge rectangular extret voltant de l'ull en una imatge de càmera i el punt de mirada. Aquí, fem una extensa anàlisi dels factors que afecten l'execució d'aquest seguidor en diverses configuracions experimentals, per tal d'abordar aquests problemes en futurs treballs. A la segona part del nostre treball, proposem un mètode d'estimació de la mirada basat en característiques, que codifica la imatge de la regió ocular en una representació compacta. Argumentem que aquest tipus de representació és més adequat per a tractar amb la postura del cap i canvis en la condició d'il·luminació, ja que redueix la dimensionalitat de l'entrada (és a dir, la imatge de l'ull) i trenca la connexió directa entre les intensitats dels píxels de la imatge i l'estimació de la mirada. Finalment, utilitzem un algoritme d'alineació de la cara per tenir una estimació de la postura de cara robusta, fent servir un model 3D personalitzat per al subjecte que fa servir el seguidor. Combinem això amb una xarxa neuronal convolucional entrenada en un gran conjunt de dades d'imatges per aconseguir un seguidor de mirades invariant a la postura de la cara.

Paraules clau: *seguiment dels ulls, seguiment de la mirada, interacció de l'ordinador humana, visió per computador*

Contents

List of Figures

List of Tables

1 Introduction

From a computer scientist's aspect, human beings are machines which receive input from their sensors such as ears, eyes, skin and which interact with the world they live in through their actuators, which are their hands, feet, and so on. Therefore, as in the case of robots, one can understand the basis of their reasoning by inspecting the input they receive and also how they direct the attention of their sensors, for instance by looking at specific locations or inspecting unknown objects by touching or smelling.

Eye-tracking studies build upon this insight and analyze the relation between the movements of a person's eyes and their attention, reasoning and feelings. These studies make use of special equipment called eye-trackers, which either calculate the direction of a person's gaze [133] or a point in the surrounding area where the gaze is fixed at. This point may be located on a 2D plane [15] (i.e. the display of an electronic device) or a 3D volume [75] and is also called the point of regard (PoR) [33]. There exist several commercial models of eye-trackers [5, 29] which come with their software platforms to carry out eye-tracking research, also expertise for calibration and actual usage is provided through documentation and support.

Although the commercial components provide satisfactory performance for marketing or scientific research, the scalability of this methodology is problematic because these products require trained operators and their price (starting from hundreds of Euros) makes them not suitable for this purpose. Therefore, the availability of a cheap, easy to setup alternative which does not require special hardware and which provides comparable performance is a necessity.

Building this alternative constitutes the problem we address in this work, and it consists of understanding where a subject is looking at using cheap components such as light sources, cameras and a computer to run the eye-tracker software. Actually, these requirements are already met in many consumer electronic devices such as laptops, smart-phones, tablet computers. We believe that the availability of a system running on these machines will provide basis for a variety of applications. For example, the eye-tracker can be used as a new input method for the electronic device just like a mouse [54, 113], or it can be used as a tool to enable remote

marketing and usability research [2, 46]. The system can replace commercial eye-tracking equipment to the extent allowed by its performance, and it can enable mobile applications of these techniques. The uses of such a system is only limited by the imagination of software developers, who will surely come up with many usage areas that we don't discuss here in order not to get off our topic.

1.1 Brief information on human visual system

In order to be able to understand the methodologies behind eye-trackers, one should be familiar with the human visual system (HVS), in particular with the human eye. Figure 1.1 shows the parts of the eye which are of interest for eye-trackers. The white part of the eye which is seen from outside is called sclera, and in the front part of the eye there exists cornea which is a transparent layer. Cornea lets the light inside of the eye, which passes through an opening called pupil. Around pupil, we find the iris which is the colored part of the eye as seen from outside. Iris controls the amount of light which passes through pupil, and in a sense acts like the diaphragm of a camera. The light that enters the eye is refracted by the eye lens, and then it falls onto the retina layer which covers the inner surface of the eyeball, forming the image of what is seen through the eyes. Here lies the fovea, which is the area where eye's light sensing cells are accumulated in huge numbers. The part of the image that corresponds to this area is special, in the sense that the human visual attention is mostly directed here [20]. In the figure, two axes are shown: the optical axis passes through the eyeball center (E) and the pupil center, whereas the visual axis passes through the fovea and a special point called corneal center (C). This difference is caused by the fact that fovea does not lie on the optical axis.

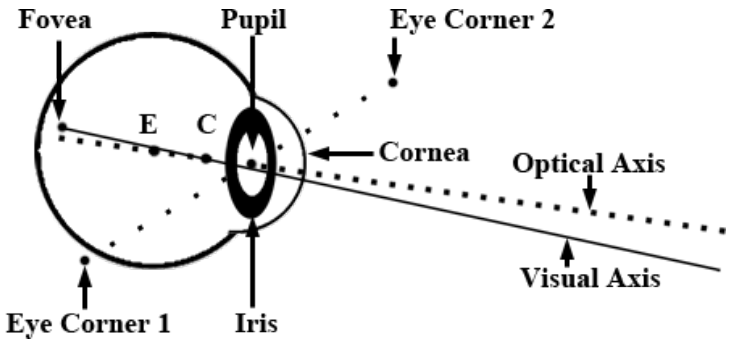


Figure 1.1 – Human gaze geometry

Two other points important for software implementations of eye-trackers are the eye corners. These are the extreme points of sclera that are visible from the outside. As the iris and pupil change location due to the rotation of the eye, the corner points become important for the software because they are mostly stable and help understand the relative movement of the eyeball.

Another important topic about HVS that remains is related to the movements of the eye. As noted before, the visual attention of human beings are mostly directed towards the objects that lie on the visual axis and whose image drops onto the fovea. Therefore, the movements of the eye should be seen as a change of attention point. The eye movements can be mainly classified into three categories [20]:

Saccades: These are rapid eye movements that are aimed at focusing the visual attention to a different object. For example, when a computer user jumps from one menu item on the display to another, the eye moves rapidly in order to focus on the next target. The duration of these movements is between 10 ms and 100 ms, and during this short period the visual system becomes practically blind as the attention cannot be directed at a particular object.

Smooth Pursuits: These happen when the eyes are following a moving target. In order to achieve this movement, the speed of the target should be in the range that can be handled by the human eye.

Fixations: These are the movements which stabilize the retina on the object of interest. This is particularly interesting for our work because during fixations, the subject directs their attention on the object that lies on the visual axis and calculating the gaze point makes sense.

Apart from these main movements, blinks which are involuntary movements to keep the eyes moist, cool and clean [33] should be considered. For the correct operation of an eye-tracking system, these movements should be detected and excluded from training and testing processes.

1.2 Objectives and Scope

The aim of this PhD dissertation is to investigate methods to estimate a person's gaze with the help of computer vision and machine learning, by developing a software that can run on any computing device with a front-facing camera. This requirement is already satisfied in almost all commercial personal computers, mobile phones, and tablet devices available for sale at this time; and a solution that can run real-time with the computational power of these devices would open many new possibilities in the area of human-computer interaction (HCI).

With this goal in mind, we first focus on assessing the **limits** of a gaze tracker and the **factors** that may affect its performance, asking questions such as:

- How does the camera placement affect the accuracy of a gaze tracker?
- Does the accuracy change in mobile environments, where the viewing distance and angle is different compared to desktop setups?
- Is the estimation robust to personal physical differences? Can it fail to provide decent estimations for everybody?
- Can head movements be handled easily by these systems?

In the first part of this thesis, we propose our first solution, and run thorough tests to search for the answers to these questions. For these experiments, we collect a dataset of 48 videos from 12 participants, and make it publicly available for fellow researchers. Here, we also describe a novel gaze estimation method, which makes use of features extracted through the segmentation of iris region. By investigating this idea further, we search for the answers to:

- How can we encode the information of eye appearance into a compact representation, which can later be used in predicting the gaze?
- Is there an middle-ground between the two extremes of *a)* using the image pixels directly (around 1k features, most of which are irrelevant) and *b)* summarizing the information too much and using iris center position (2 features) or iris position & normal angle (6 features)?
- Does removing the direct dependency to image pixel intensities improve estimation accuracy, removing the effects of different lighting conditions?

The second part of this work is aimed at making use of large datasets (>100k samples) made available recently. We experiment with several convolutional neural network (CNN) architectures that can harness the potential of these datasets, and compare the results with traditional gaze tracking methods. With this work, we hope to answer the following questions:

- Can pouring more data into gaze estimation methods improve the gaze estimation performance and robustness?
- Which methods are more suitable to learn from more data?
- Do CNN architectures work well with very small images (an order of magnitude less pixels than ImageNet samples) that are commonly processed by gaze estimation systems?

Finally, the third part of our work is aimed at building a complete gaze–tracking system around the work done on the software implementation. This system includes a hardware prototype on which we run our eye–tracking software, and an example use–case for human–computer interaction.

The objective of this PhD is to shed light on the answers to all these questions, with the aim of building a gaze tracking system that can learn from data to the maximum extent, providing a performance comparable to commercial eye tracker machines.

1.3 Outline

This PhD thesis has been divided into four parts. In the next chapter, we explain the state-of-the-art in the field, thoroughly analyzing all the works from several aspects such as head pose invariance, calibration strategies and gaze estimation methods. We make a comparison of the accuracies, the strengths and weaknesses of each method, in order to provide a map where our work can be contextualized correctly. Moreover, we review developments in related topics for natural light gaze tracking, such as publicly available datasets, open source projects, commerial solutions providing gaze tracking as a service, etc.

Chapter 3 starts with our initial investigation about the limits of natural light gaze trackers, assessing the affects of several factors such as camera positioning and resolution, head stability, etc. in their performance. Here, we propose our first complete system which provides a semi-automatic initialization procedure that acts as an initial step to calibration. The second part of Chapter 3 describes a new method for eye feature representation, which can be mapped to the gaze coordinates with the help of a regressor. This method is in contrast with traditionally used representations, where the most common options are using the image pixel intensities directly or summarizing the eye appearance with the iris center location and/or the iris normal vector.

In Chapter 4, we investigate the methods made possible by the recent availability of large-scale datasets (>100k samples). In other computer vision areas such as object detection, larger datasets have been continuously helping improve the state-of-the-art in the recent years, as they provide enough learning material to train more and more complex models. In this part of the thesis, we use a publicly available dataset to train different convolutional neural network (CNN) architectures, in order to find a model which can learn as much as possible from this large dataset.

Chapter 5 discusses what we call *our System*, which consists of the software structure as described in the first chapters, a prototype for a cheap hardware structure, and an example use–case. Here, we also explain the performance improvements

that we achieve after parallelization of two key components of the software modules.

Finally, we wrap up this thesis in Chapter 6, going over our contributions and listing the outcomes of these works.

2 State-of-the-Art in Natural Light Gaze Tracking

Currently, the most widespread techniques used in eye trackers make use of light sources and cameras that operate in the infrared (IR) spectrum. There are many available commercial models that are either in the form of eyeglasses or table mounted devices [4, 5, 29] and also open source alternatives that allow the use of custom hardware [88].

Visible light gaze tracking, on the other hand, does not require any special hardware and aims to solve the task making use of regular cameras. In this thesis, we concentrate our work on this class of trackers and this chapter includes a survey of the related research. Furthermore, we will limit our search to the table mounted setup (also called remote setup) as it is ubiquitous in contemporary devices and it removes the restrictions for camera placement (with a few exceptions). The aim of the review in this chapter is:

- Provide an exhaustive literature review
- Comment on these works from various perspectives
- List publicly available datasets
- List open source software
- List gaze trackers as a web service

The rest of the chapter is organized as follows: we will start with an overview of the software structure used in remote, visible light gaze trackers. Then, we will categorize and explain the previous work according to the techniques used, and continue with two other categorization schemes: how/if they are calibrated and how/if they handle head movements. Afterwards, we will list and comment on the available datasets, online gaze tracking services and open source projects. We will finish with our conclusions regarding the current state and future directions.

2.1 Categorization and Structure of Visible Light Gaze Trackers

The categorization of the works that we analyze in this review is not trivial, because the borders between groups of methods are not always clear and in the literature different naming schemes exist.

In the early review by Morimoto and Mimica [74], methods using the eye appearance (i.e. eye region image pixels) directly for gaze estimation are called appearance-based or view-based methods, and the rest is left unnamed. Here, the given name refers to all the visible light methods and does not give information about the sub-categories. Even in a more recent survey [33] where both infrared (IR) and visible light methods are considered, the latter group is considered as just an alternative, and its sub-categories are left unclear. Other categorization schemes also build on this ambiguity: appearance-based vs. feature-based [60, 64], appearance-based vs. model-based [24, 101]. It should also be noted that the "appearance-based" name is still being used to refer to all visible light methods [61, 131], adding to the confusion.

With the aim of clearly identifying the borders between different visible light gaze estimation techniques (and hopefully not adding to the confusion), we present a new categorization scheme:

a) **Appearance-based:** These methods only use the eye image pixel intensities to create a mapping to the gaze estimation. The image pixels are converted to a vector representation via raster scanning and fed to the estimation component.

b) **Feature-based:** Methods of this category also make use of a mapping to calculate the gaze; however, they use richer feature vectors compared to the methods in the previous category (i.e. not just pixel intensities).

c) **Model-based**: Compared to the discriminative approach of the first two categories, the methods belonging to this category follow a generative approach by trying to model the eyes and maybe even the face. The gaze is calculated geometrically using the model parameters.

After explaining our categorization and the reasoning behind it, we can continue with the discussion about the software pipeline of these trackers. Although the variation in details are huge, a common skeletal structure that describes their software implementation can easily be identified as seen in Fig. 2.1.

The input to the system is generally a video stream; however, examples of systems working on still images are also found [144]. In the former case, the previously

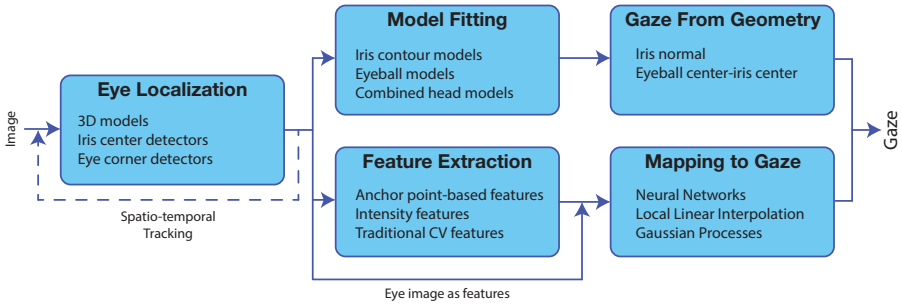


Figure 2.1 – The common software structure for visible light gaze trackers. The methods start by locating the eyes. To make the estimation more stable, spatio-temporal tracking may be utilized at this step. Later, the location information is used to extract features, fit 2D or 3D models to the eyes or just to extract the eye region image. In the case of model-based methods, the fitted model is used to calculate the gaze geometrically; whereas in the other methods a mapping function is necessary to calculate the gaze angle or point.

processed video frames' results may be used to improve the performance for the next frames [127].

The first task in the pipeline is to extract the eye region. If an optional head pose estimation component is present, and if its output contains information about the eye location; it may be used directly as the location or it may be used as a rough initial estimate for the actual eye locator. Otherwise, the eye locator component has the option of using face detectors to restrict the processed image area and reduce computational cost [100, 119]. In order to calculate accurate eye location, the system can make use of iris center detectors [95], eye corner detectors [72] or 3D eye models that take into account the appearance of the entire eye [141].

Once the region of interest (ROI), that is the eye region, is located; the second step is to prepare the input for the gaze estimation component. Depending on the class of gaze estimation method, the required input for the last step varies. In **appearance-based** methods, the extracted eye image from the first step is used directly as the input. Here, each image pixel intensity is considered as one dimension of the input vector. As the change in illumination and shadows may interfere with these inputs, this class of methods may not always give robust results.

Feature-based methods try to break the direct connection between the raw pixel intensities and the final input vector, in an attempt to increase robustness to lighting changes. Some of the features used in the literature are:

- Pixel positions of keypoints (e.g. inner eye corners, iris center, eyelid) [10, 32]
- Their relative positions (i.e. vectors connecting two positions) [112, 115, 147]
- Standard computer vision features such as histogram of oriented gradients (HOG) [18, 70], local binary patterns (LBP) [61, 81]
- Features calculated by a convolutional neural network (CNN) [144]
- Features grouping and summarizing pixel intensities [21, 65, 66, 138]

Finally, the **model-based** gaze estimation methods require the parameters for a 2D or 3D eye model as the input. In case of 2D, these can be the parameters defining the iris edge model [121]; in the 3D case it can get more complex to include 3D positions of the eyeball center [15] or other facial landmarks [50].

The last step in the described pipeline is the estimation of the gaze, given the inputs calculated in the previous step. Appearance-based and feature-based methods require a mapping function that maps the input vectors to the gaze point or the gaze direction. The commonly used techniques include: neural networks (NN) [12, 146], Gaussian process (GP) regression [78, 127] and linear interpolation [91, 107], among others. On the other hand, model-based methods use the geometry of their 3D model (e.g. normal vector for the iris of 3D eye ball model) to calculate the gaze [87, 134].

2.2 Methods for Single Camera Remote Gaze Tracking

In this section, we categorize the works that we focused on according to our scheme. A summary of these works can be seen in Table 2.1, Table 2.2 and Table 2.3.

2.2.1 Appearance-Based Methods

The first techniques proposed for visible light gaze tracking introduced the category of appearance-based methods [12, 100, 137]. These methods are characterized by their use of eye image pixel intensities as their features for gaze estimations. After a possible histogram normalization step for standardizing image appearances over the whole dataset, these feature vectors are fed to the estimation component which maps them to screen coordinates.

Neural Networks

One of the most popular mapping functions used in eye tracking is neural networks (NN). In their pioneering work, Baluja and Pomerleau [12] introduce the first

Table 2.1 – Summary of appearance–based methods. Calibration column denotes how/if the system is calibrated. HP column shows whether the technique has head pose invariance or not. Techniques allowing small head movements are denoted by the ≈ symbol. Accuracy column shows the reported error in degrees.

Mapping	Calibration	HP	Acc.	References	Comments
NN	Grid	—	1.5–4	[12, 40, 92, 100, 137]	
GP	Grid	—	2	[24]	
GP	Grid	≈	n/a	[77, 78]	Rigorous calib. for HP
LLI	Grid	—	0.4	[107]	IR to locate eye
LLI	Grid	—	2.4	[82]	
LLI	Grid + HP	✓	2.2–2.5	[64, 65, 67, 69]	0.85° w/ fixed HP
LLI	Grid	✓	4.8	[60]	
LLI	—	✓	3–5	[104, 105]	Incremental calibration
LLI	Grid	✓	4	[102]	8 cameras
LLI	—	—	3.5–4.3	[7, 101]	Saliency for calibration
LLI	Grid	✓	3.4	[142]	0.9° w/ fixed HP
LLI	—	✓	9.9	[129]	Synth. training data

method making use of NNs. They test their system extensively by varying the inputs (iris region or entire eye), NN structure (single continuous or divided hidden layer) and the hidden layer unit number. In another experiment, they demonstrate that by training the system with inputs from different head poses, the system can even handle small head movements. Finally, they top their system with an offset table that is used to correct the systematic shifts in actual eye tracker use. In the best case, their reported accuracy is around 1.5°.

Stiefelhagen et al. [100] use skin color segmentation and pupil detection to replace the use of a light source for this task in the original Baluja and Pomerlau work. Xu et al. [137] introduces an iterative thresholding method to locate the iris region accurately, and also propose a Gaussian smoothing for outputs of the NN during training. Two recent works [40, 92] used the NN technique for gaze tracking on commercial tablet computers and report lower accuracy (average error > 3°), mainly because of the low sampling rates in tablets and high training data demand of the NNs.

Local Linear Interpolation

A recently more popular alternative to NN mapping is local linear interpolation as proposed for gaze tracking by Tan et al. [107]. In their work, they see the eye region images as coming from an appearance manifold, and gaze estimation is posed as a linear interpolation problem using the most similar samples from this manifold. Although this work makes use of IR illumination for eye localization,

Table 2.2 – Summary of feature–based methods.

Feature	Mapping	Calibration	HP	Acc.	References	Comments
PC-EC	GP	Grid	—	1.6	[32, 34]	
PC-EC	LI	Grid	—	1—1.2	[135, 147]	
PC-EC	LI	Grid	—	n/a	[115, 117]	
PC-EC	PI	Grid	—	3	[91]	1.2° w/ chinrest
PC-EC	LI	Grid	✓	2–5	[116]	
PC-EC	PI	Grid	—	2.4	[11]	
PC-EC	PI	Grid	✓	2.3	[72]	1.2° w/ fixed HP
Several	NN	Grid	—	1–2	[111, 112]	
Several	NN	Grid	✓	2–7	[10]	Few tests
EC shift	n/a	Grid	—	3.2	[48]	
EC shift	LI	—	—	3.4	[79]	Hand-coded params.
GC-CM	LI	Grid	—	1.5	[128]	
Several	LI	Grid	—	3	[95]	
Edge energy	S3GP	Grid	—	0.8	[127]	
Intensity	ALR	Grid	≈	0.6	[66, 68]	8D or 15D feats.
Intensity	RR	Grid	—	1.1	[138]	120D feats.
HOG	SVR/RVR	Grid	—	2.2	[70]	
Several	NN	Grid	—	3.7	[146]	Dim. reduced to 50
CS-LBP	S3GP	Grid	—	0.9	[61]	Partially labelled data
Several	Several	Grid	—	2.7	[90]	Dim. reduced to 16
Several	Several	Grid	✓	3.2	[43]	
Segmentation	GP	Grid	—	2.2	[21]	
CNN	Several	—	✓	~6	[144]	Calib. from dataset
CNN	CNN	—	✓	7.9	[130]	Calib. from dataset
CNN	CNN	—	✓	1.7cm	[57]	Calib. from dataset
Img. patches	SVM	—	—	7.5	[63]	Posed as classification
BRIEF	SVR	Grid	✓	4.1	[51]	

the gaze estimation technique is valid for purely visible light setups. The reported accuracy of around 0.40° shows the promise of the proposed technique.

Ono et al. [82] calculate the decomposition of the eye image, which takes into account variations caused by gaze direction, base eye appearance and shifts in image cropping. Using this decomposition, they can encounter the most similar 3 training samples and use LLI to calculate the gaze with 2.4° accuracy.

Sugano et al. [104] use an LLI technique that allows head movements. They cluster the eye images according to the corresponding head pose and choose samples for interpolation only from the cluster with the same head pose as the current sample. Their system keeps learning from user interaction (i.e. mouse clicks) and continuously updates its parameters, adding clusters for new head poses when necessary. The reported average error is between $4-5°$. The extended version of the work [105] provides methods for refining gaze labels acquired through mouse clicks, discarding high-error training samples and locating the eye position better;

Table 2.3 – Summary of model–based methods.

Model	Calibration	HP	Accuracy	References	Comments
Iris contour	Camera	✓	1	[121, 122]	One-circle alg.
Iris contour	Grid	✓	4	[35, 36]	
Iris contour	—	✓	n/a	[133]	Two-circle alg.
Iris contour	Camera	—	n/a	[44]	
Iris contour	Camera	—	0.8	[143]	Error for single dir.
Iris contour	Grid	✓	3.3	[26]	
Iris contour	Grid	✓	3.5	[73]	
Iris contour	Grid	✓	6.9	[131]	
Eyeball	Grid	✓	3.2	[50]	Calib. personal params.
Eyeball	Grid	—	3.5	[134]	PF tracking
Eyeball	1 target	✓	~2	[1]	Error for single dir.
Eyeball	Grid	✓	2.7	[15]	
Eyeball	—	✓	9	[140, 141]	Auto calibration
Eyeball	Grid	✓	n/a	[87]	
Eyeball	—	✓	5.6	[39]	
Eyeball	—	✓	8	[17]	3.8° w/ fixed HP

thus decreasing the average error to only 2.9°.

Lu et al. [64, 65] decompose the gaze estimation problem into subproblems: 1) estimation under fixed head pose, 2) compensation of errors caused by head rotation and eye appearance distortion. Unlike other works, they do not choose most similar local training samples explicitly; however, they argue that their method for weighting all the training samples automatically selects a small number of local samples. By learning eye appearance distortion from 5-second video clips and applying both compensations, they decrease the average error from 6° to 2.38° (and from 13.72° to 2.11° in the 2014 paper). In their later work [67, 69], instead of video clips (containing around 100 frames), they acquire only 4 additional training samples under reference head poses and synthesize extra training samples by modeling the change in eye appearance.

Alnajar et al. [7] propose a calibration-free estimation based on the assumption that humans have similar gaze patterns for the same stimulus. Here, first initial gaze points are calculated for a user without calibration, then a transformation is calculated to map the user's gaze pattern to other users'. For the initial gaze estimation, they either use the closest neighbors from the training set to reconstruct the current eye appearance (with samples from other users) or project the eye appearance to a 2D manifold to get the most similar samples.

Lai et al. [60] use random forests to learn the neighborhood structure for their joint head pose and eye appearance feature (HPEA). Gaze is estimated with linear interpolation using the neighbors in the random forest, yielding an accuracy of

around 4.8° (horizontal and vertical combined).

Sugano et al. [102] build a multi-view dataset and use it to reconstruct part of the face in 3D. They use this 3D model to generate synthetic samples acquired from different camera angles and use the extended dataset to train a random forest. Here, unlike their previous work [104], they do not divide the data strictly according to the head pose; however, they build sets of regression trees with overlapping head pose ranges (i.e. samples from a single head pose is used in building several sets of trees). Gaze is calculated as the average result from the nearest regression forests according to head pose, resulting in an average error of 6.5° with cross-subject training.

Wood et al. [129] also take the route of generating synthetic data for training, using a generative eye region model to render realistic eye images. In a cross dataset experiment, they achieve 9.9° gaze errors.

Yu et al. [142] argue that in local linear interpolation, the common practice of using the same reconstruction weights in the appearance space for gaze estimation is flawed, and they propose an adjustment to address this issue. This technique reduces the error rates by 12—63%, yielding error rates of 0.9—6.1°, depending on whether there is head pose and subject variation.

Gaussian Processes

Gaussian Process (GP) is another choice for the mapping in some gaze tracking methods. GP predictions are probabilistic, and allow calculation of confidence intervals for the outputs which may be used as an indicator to detect when the calibration is no longer valid for the test data [32, 86].

Nguyen et al. [77, 78] describe a system where they use a Viola-Jones [119] eye detector and optical flow (OF) to detect and track the eye in the camera image. Then, the extracted eye image is fed to a GP to calculate the gaze point. In the extended work [78], they show that when the calibration is repeated in several head poses, the system can even become head pose invariant.

Sugano et al. [101] use saliency information to automatically calibrate a gaze tracker while the subject is watching a video clip. While calibrating the GP-based tracker, instead of using known gaze positions, they train the GP with gaze probability maps calculated by aggregating several saliency maps.

2.2.2 Feature-Based Methods

In the appearance-based methods, the inputs to the mapping functions were the same across all techniques, therefore we categorized them according to the mapping functions they used. However, in feature-based methods the main difference is their feature set, and our categorization also reflects this difference.

Anchor Point Position-Based Features

In this first subcategory of feature-based methods, the positions of important anchor points inside and around the eye (e.g. pupil (iris) center, inner and outer eye corners, nostrils) are used as features. In some cases, they constitute distinct dimensions of the feature set; whereas in other cases, the relation between them (i.e. the vector connecting two anchor points) is used as the feature.

Pupil Center-Eye Corner Vector

In infrared gaze trackers, a feature widely used for gaze estimation is the pupil center-corneal reflection vector (PC-CR) [91]. The equivalent of this in natural light methods is the pupil center-eye corner vector (PC-EC) (or alternatively, iris center-eye corner (IC-EC) vector).

The first use of the PC-EC vector in natural light eye trackers is proposed by two distinct research groups around the same time [32, 34, 147]. Hansen et al. [32, 34] use Active Appearance Model (AAM) and mean-shift to track the eyes over time and find the positions of pupil center and eye corners. Gaze estimation is done by training a Gaussian Process (GP) where the input is the PC-EC vector. The system results in an average error of around 1.6°, and the eye tracker is verified in an eye-typing interface. Zhu and Yang [147], on the other hand, propose methods for detecting the iris center and the eye corner with subpixel accuracy. They use a 2D linear mapping to estimate gaze positions from the feature vectors. They report an accuracy of around 1.2° from their experiments.

Valenti et al. [115, 117] propose a novel eye corner locator and combine it with a state-of-the-art eye center locator to calculate the EC-PC vector. Inspired by Zhu and Yang [147], they also use a 2D linear mapping for gaze estimation. In their later work [116], they make use of a head pose estimator and use the calculated transformation matrix to normalize the eye regions. The more accurate eye location found this way, in turn, is used to better estimate the head pose in a feedback loop. To solve the gaze estimation problem with head movements, they *retarget* the known calibration points to the monitor coordinates whenever there is a change in the head pose, and calibrate the system again. With these improvements, they achieve average errors of between 2° and 5° in two experimental tasks.

Sesma et al. [91] normalize the PC-EC vector, dividing the vector components by the Euclidean distance between the inner and outer eye corners. For gaze estimation, they use both PC-EC vectors for the inner and outer eye corners and their experiments show the average error to be 1.25° when the head movement is constrained, and around 3° when no chin rest is used.

Baek et al. [11] apply image rectification to rectify the eye images to a front facing head pose, and combine it with a novel iris center localization method. They use second order polynomial equations (as in [91]) to calculate the gaze and measure

an accuracy of 2.42°.

Cheung et al. [72] fit Active Shape Models (ASM) on images normalized using local sensitive histograms. With the novel methods they propose for iris center and eye corner detection, they achieve errors of 1.28° with fixed head pose and 2.27° with head movements.

Others

Some feature-based methods making use of anchor point positions may take a different path, and combine or replace the EC and PC positions with information coming from other anchor points (e.g. nostrils) or simply calculate the features in another way.

In his thesis, Bäck [10] uses several geometrical features such as iris center, eye corner, nostril positions, head angle, eye angles to create a rich feature vector, and trains a NN for gaze estimation. The system is not tested heavily; however, the accuracy is reported to be between $2-4°$ and sometimes even up to $7-8°$.

Torricelli et al. [111, 112] calculate several distance and angle features from both eyes to fill the feature vector. These features include distances of inner and outer eye corners to the iris center, the slopes of the lines connecting these points and the positions of outer eye corners. The trained NN gaze estimation component results in average errors between $1-2°$.

Ince and Kim [48] track the iris with a custom method, and calculate the gaze using the iris center displacements between subsequent camera frames. The proposed system has an accuracy of 3.23° (horizontal and vertical combined). Nguyen et al. [79] take a similar approach and make use of the center-bias effect, which states that gaze distribution is biased towards the center of the screen [52]. Their system does not require any calibration, and works by calculating the mean iris center over time and estimating the gaze through the difference of current iris center and the mean. The combined error in x and y directions are 3.43° of visual angle.

Wojciechowski and Fornalczyk [128] preprocess the eye images by calculating the edges, and then extract their features which are the geometric center and center of mass of edge pixel positions. The final feature is the vector connecting these two locations (GC-CM), which is used to calculate the gaze estimation using the weighted average of data from 4 training points. The system has around 1.5° accuracy (combined).

Skodras et al. [95] track several moving and stationary anchor points (e.g. eye corner, eyelid control points, iris center) and calculate vectors from their relative positions to build the final feature vector. They use linear regression for mapping to gaze point and achieve an accuracy of 2.96° (combined).

Intensity-Based Features

In some feature-based methods, the direct connection between the image pixel intensity and feature vector is not broken completely. Williams et al. [127] combine the image pixel intensities with edge energies in their feature vector. They train a sparse, semi-supervised Gaussian process (S^3GP) which also infers the missing labels in the partially labeled training data. They make use of the confidence value for the GP to filter the estimation over time using a Kalman filter and achieve a final accuracy of 0.83°.

Lu et al. [66, 68] propose extracting 8D or 15D intensity features from the eye region, which is identical to resizing the grayscale eye image to 2×4 or 3×5 pixels, respectively. Together with the proposed subpixel alignment method for eye region, and adaptive linear regression (ALR) for gaze estimation, they can estimate the gaze point with up to 0.62° of accuracy.

Xu et al. [138] extend the work of Lu et al. [66, 68] to increase the feature dimension to 120D (2 eye images of 6×10 pixels) and to use ridge regression for gaze estimation, and achieve slightly worse results (1.06°).

Traditional Computer Vision Features

Computer vision (CV) tasks such as object detection, classification, etc. are normally solved by using features (e.g. histogram of oriented gradients (HOG) [18], scale-invariant feature transform (SIFT) [62], local binary patterns (LBP) [81], etc.) extracted around salient points in the images. However, until recently, this approach was still unexplored for the gaze tracking problem.

Martinez et al. [70] introduce this concept in a head mounted tracker, where they extract multilevel HOG features from eye images and use support vector regression (SVR) or relevance vector regression (RVR) to map these features to the gaze point, and achieve an accuracy of 2.20° (combined).

Zhang et al. [146] combine several features to build their feature vectors: color, pixel intensity, orientation (from several Gabor filters), Haar-like features, spatiogram features (combining color histogram with spatial information). After generating this rich representation, they apply a dimensionality reduction technique to reduce the feature vector size to 50, and train a NN for gaze estimation. Although the reported average error is not very low (around 3.70°, when combined), the work is a great example of applying the traditional CV pipeline to gaze trackers.

Liang et al. [61] build on the previously explained S^3GP technique [127], and train it with CS-LBP features [38], which is based on LBPs. They make use spectral clustering to learn from partially labeled data, and report an average error of 0.92°.

Schneider et al. [90] explore several feature types (DCT, LBP, HOG) in conjunction with many alternatives for regression (GP, k-nearest neighbors (kNN), regres-

sion trees, SVR, RVR, splines). They use a *dually supervised embedding* method to reduce the feature dimensionality, resulting in up to 31.2% decrease in the errors (best accuracy being 2.69° with 16-dimensional features based on HOG and LBP). Huang et al. [43] also take the same approach and review several feature types (LOG, LBP, HOG, mHOG) and regression components (kNN, RF, GP, SVR). They report that random forests (RF) combined with multilevel HOG (mHOG) features prove to be the most efficient combination (3.17° error) in a very challenging scenario (i.e. tablet computers), with free head movements.

Jeni et al. [51] make use of BRIEF binary features [13] extracted around the eye and calculate the gaze estimation using SVR. The reported accuracy is between 3.9—4.1°, which drops to 5.9° for lower resolution images.

Lu et al. [63] build a "codebook" (or vocabulary) of image patches, and calculate the reconstruction of eye images using a combination of these patches. The representation of this reconstruction is used as the feature set for a multi–class linear SVM, where the final error rate is around 7.5° on average.

Lately, convolutional neural networks (CNNs) are very popular in computer vision research, and Zhang et al. [144] are the first to use it for gaze tracking. CNN methods generally require a large dataset, and in their work they also present their dataset [145] which contains more than 200,000 images. They calculate features using a CNN, and combine it with the head pose information to build the complete feature vector. After testing with several regression functions (random forests, kNN, ALR, SVR), the best accuracy they achieve is around 6°.

Wood et al. [130] feed synthetic images together with another synthetically augmented dataset [103] to a CNN, and reach 7.9° cross–dataset average error rates. The latest example of using CNN architectures for gaze estimation is from Krafka et al. [57], where they enrich the input by using eye region images in conjunction with head region images and a grid representing the location of the head inside the camera frame. The average error of their system is around 1.04cm on mobile phones, and 1.69cm on tablets.

2.2.3 Model-Based Methods

The models used in model-based gaze estimation methods are roughly divided into two: iris contour models (also known as one-circle algorithm) where an ellipse is fitted around the iris region, and eyeball models where the main objective is to estimate the location of the eyeball center.

Iris Contour Models

The direct least squares method for fitting ellipses onto a set of points [25] is influential in the development of iris contour models for gaze estimation. This method, complemented with the observation that the circular iris boundary appears as an ellipse in camera images, has enabled the development of several gaze tracking techniques.

Wang et al. [121, 122] develop the one-circle algorithm where they use edge detection to find pixels belonging to the iris boundary, and they fit an ellipse to this set of locations. Then, the ellipse is back-projected to the 3D space to find the iris contour circle, and its normal vector is used as the gaze vector. Their system has an average error of around 1°.

Hansen and Pece [35, 36] use an active contour method to track the iris edges over time, and (probably) using the one-circle method, their system estimates the gaze with around 4° accuracy.

Wu et al. [133] propose an extension with their two-circle algorithm, where they assume the elliptic iris contours for both eyes lie on the same plane or on parallel planes in 3D. With this assumption, they are able to estimate the gaze direction without the need for camera calibration.

Huang et al. [44] use randomized Hough transformation for iris contour fitting, whereas Zhang et al. [143] propose an improved RANSAC algorithm. The reported accuracy for the latter work is 0.8° in a single direction.

Fukuda et al. [26] propose subpixel methods for iris contour estimation in low resolution images, achieving a combined average error of 3.35°. Mohammadi and Raie [73] train a support vector machine (SVM) to filter out the unrelated edge segments before applying the ellipse fitting, yielding an accuracy of 3.48°.

Wood and Bulling [131] detect the edges belonging to the iris from the image's radial derivative. After fitting the ellipse using the RANSAC method, the gaze estimation has an accuracy of 6.88°.

Eyeball Models

Eyeball model-based techniques try to infer the eyeball center position, and calculate the gaze vector as the line connecting this point with the iris center.

Ishikawa et al. [50] use an AAM to track the face, and use the eye corner positions and the scale of the face to infer the anatomical constants for the user (i.e. eye geometry). This calibration is followed by iris detection by template matching and edge-based iris refinement to calculate the center of the iris. The geometrically calculated gaze has an average error of 3.2°.

Wu et al. [134] track the iris contours and the eyelids with a particle filter (PF), and use several appearance metrics to calculate the likelihood of a given particle

(candidate). Experimental results show the mean error to be greater than 3.5°.

Xie and Lin [1] infer the position of the eyeball center and other personal parameters using a simple one target calibration. They calculate the gaze geometrically by using the iris center and eye corner positions on the image, with 2° accuracy in a single direction.

Chen and Ji [15] use a generic face model that includes several facial anchor points (nostrils, inner and outer eye corners) and one of the eyeball centers. After calibrating for the personal parameters, they track the facial points and fit the 3D model to estimate the gaze with 2.7° accuracy.

Yamazoe et al. [140, 141] segment the eye image pixels into three classes: skin, sclera, iris. Using the segmentation results, they calculate the most possible eye pose by minimizing the projection errors for a given candidate. The accuracy of the system is reported to be around 9°.

Reale et al. [87] use the detected iris contours to calculate the eyeball center, and after calibrating for the visual axis-optical axis shift and the eyeball radius, they estimate the gaze direction. Finally, the most recent work in this category is from Heyman et al. [39], who employ canonical correlation analysis (CCA) to estimate the head pose in a similar manner to AAMs. They calibrate the eyeball radius during initialization, and estimate the iris center using a segmentation method. Their system estimates the gaze direction with 5.64° accuracy.

Cristina and Camilleri [17] use a similar cylindrical head model as in [116], combined with spherical eyeball models. Their method has 3.84° accuracy with little head movement, which drops to 8° with free head movement.

2.3 Calibration Strategies

Traditionally, calibration of the eye trackers consists of asking the subject to look at several targets in known positions. This way, either the personal parameters (e.g. angle between visual and optical axis of the eye, eyeball radius) or the camera parameters (e.g. focal length, position with respect to the display) are learned.

Several papers that we analyze in this review present novel techniques to make this process easier for the subject using the tracker. Yamazoe et al. [140, 141] employ a transparent calibration process, where the user does not need to be aware at all. They track the face over time to construct the 3D model of the face and eyes, and start calculating the gaze when the calibration is ready. Alnajar et al. [7] use other users' gaze patterns to help estimate the current user's. Sugano et al. [101] completely remove the need for training data, and estimate the gaze in a probabilistic manner using computed saliency maps.

Another approach to collecting the training data without needing special actions

from the user, is to let the user operate the computer normally and take samples during mouse clicks [104, 105, 144]. This method is based on the assumption that the user looks around the mouse pointer while clicking.

Head movements constitute a challenge for eye tracker calibration, and even small movements may cause large errors in the estimations of a calibrated tracker. This holds true especially for appearance-based gaze trackers. Valenti et al. [116] solve this problem by *retargeting* the calibration targets' positions to user's new field of view, and calibrating the system again. Lu et al. [64, 65] require the user to record 5 second video clips while moving her/his head, and use these to correct errors caused by head movements. Xie and Lin [1] require just a single target calibration, where the user keeps looking at the same position on the screen and moves her/his head around. Zhang et al. [144] take an approach based on large datasets, and use other people's training data to calibrate a more accurate tracker.

Making the calibration process transparent for the user and collecting the required large amount of data are two conflicting objectives. In order to use the available training data to full extent, Williams et al. and Liang et al. [61, 127] use partially labeled data and annotate some of the unlabeled samples automatically. Ono et al. [82] create new samples by adding shifts while cropping the eye images, and this way they can model the resulting appearance change and compensate for it while searching local samples. Lu et al. [67, 69] create synthetic training data by modeling the pixel flow around the eyes, whereas Sugano et al. [102] use 8 cameras to model a large part of the face in 3D and to generate training samples from previously unobserved head poses.

2.4 Dealing with Head Pose

Model-based visible light gaze tracking methods are normally invariant to head movements, assuming the preprocessing steps such as eye localization or model fitting do not fail. However, the same does not hold for the appearance-based and feature-based systems. As Lu et al. [65] demonstrate, the head movement not only adds a shift to the gaze angle, but also makes the calibration invalid by distorting the eye appearance for appearance-based methods.

The naive approach to solving the problem of head movements is adding more training data. Nguyen et al. [77, 78] propose repeating the calibration up to 10 times, while Lai et al. [60] require 34,000 training samples per user.

Zhang et al. [144] use a large dataset of previously collected images to train a feature-based gaze tracker. Here, training data collected from many subjects can be used in estimating the gaze for another person. Head pose invariance is achieved by incorporating the head pose angles into the feature set.

Table 2.4 – Publicly available datasets for remote, natural light gaze tracking

	# Subjects	# Targets	# Head Poses	Resolution	Dataset Size	References
UUlm	20	2-9	19	1600 × 1200	2,200 imgs	[125, 126]
HPEG	10	Cont.	2	640 × 480	20 videos	[8, 9]
Gi4E	103	12	1	800 × 600	1,236 imgs	[83, 84, 118]
CAVE	56	21	5	5184 × 3456	5,880 imgs	[96, 97]
CVC	12	12-15	4	1280 × 720	48 videos	[23, 24]
EYEDIAP	16	Cont.	Cont.	1920 × 1080	94 videos	[27, 28]
Multi-view	50	160	8 (+synt.)	1280 × 1024	64,000 imgs	[102, 103]
MPIIGaze	15	Cont.	Cont.	1280 × 720	213,659 imgs	[144, 145]
OMEG	50	10	Cont.	1280 × 1024	44,827 imgs	[37]
TabletGaze	51	35	Cont.	1280 × 720	816 videos	[42, 43]

In other approaches [104, 105], the multi-pose training data is grouped according to head pose, and only a subset corresponding to the most similar head pose is used in the active calibration. To reduce the need for additional training data, Lu et al. [67] synthetically generate training samples for unseen head poses.

Instead of pouring more data into the system, another option is to apply compensations, or small fixes to keep the current calibration working. Lu et al. [68] propose an eye image alignment scheme to undo the deformation in these images. In their other works [64, 65], they train a regression for this task, and combine it with a compensation for head rotation.

Valenti et al. [116] keep the calibration targets in a flexible representation, and *retarget* these to the display coordinates whenever the head pose is changed and recalibrate their system.

Cheung et al. [72] assume the PC-EC feature is completely invariant to head pose, and apply only head rotation compensation in their system.

2.5 Available Datasets

Several papers that we analyzed contain a summary of publicly available datasets for visible light gaze tracking [96, 102, 144]. However, they are mostly for the purpose of comparison with the presented datasets in the mentioned work, and thus may lack some pieces of related information.

In Table 2.4, we bring together all the datasets mentioned in these works (with several more recently published additions), in an attempt to provide a reference for future research in the field.

One of the datasets [71] cited in the previous reviews has been removed, as it

provided data for a head mounted setup.

2.6 Gaze Tracking as a Service

While visible light gaze tracking has become a hot topic in the academia in recent years (as can be observed in Fig. 2.2), the industry is not trailing far behind either. Here, we talk about several companies already providing gaze tracking service based on regular cameras found on consumer devices.

GazeHawk [46] (now closed), was enabling its customers to convey remote eye tracking studies inside the user's browser. xLabs [136] is another similar service, which is also available as a Chrome extension. With the extension, several demos (including continuous calibration by an ant smashing game) can be tried. Lastly, Sticky [99] also provides a JavaScript-based service, suggesting use cases such as online ad placement, web page optimization, etc. As the only service with detailed specifications, their eye tracker provides an average accuracy of 2.4°.

Other possible clients for this type of eye trackers are the game or application developers. SentiGaze [76] provides an SDK for developers targeting the Windows platform. FaceTrack from Visage Technologies [108] provide a similar C++ SDK for developers, with augmented reality, view control in gaming, view-dependent rendering suggested as possible use cases. The SDK provides detailed information such as mouth contour, chin pose, eye openness, etc., in addition to the gaze information. InSight SDK [93] takes one step further and combines the gaze information with mood, age and gender estimation.

With the transition from desktop programs to mobile apps in recent years, two companies see a possibility for gaze tracking on this platform. Snapdragon [47] provides an SDK for Android apps, whereas Umoove [114] has a product on both iOS and Android platforms.

2.7 Open Source Projects

A few works that we analyze in this review have released their source code with an open source license. In this section, we list these options so that new projects in the field will have a starting point for the codebase. Table 2.5 shows a summary of the listed projects.

Opengazer [148] is an eye tracker from Cambridge University, that is unfortunately no longer maintained. It uses Gaussian process regression with eye images as features, which is similar to the technique described by Nguyen et al. [78]. NetGazer [3] is the port of Opengazer for the Windows platform and is not maintained any-

Table 2.5 – Open source gaze trackers and the related publications

	Language	Platform	License	References
Opengazer	C/C++	Linux/Mac	GPLv2	[148]
NetGazer	C++/C#	Windows	GPLv2	[3]
CVC ET	C/C++	Linux/Mac	GPLv2	[21, 22, 24]
NNET	Objective C	iOS	GPLv3	[40, 41, 92]
EyeTab	Python/C++	Windows	MIT	[131, 132]
TurkerGaze	JavaScript	All	MIT	[138, 139]
Camgaze	Python	All	?	[120]

more either.

In the recent years, a fork of Opengazer project, named CVC Eye Tracker [22] was made available and is maintained actively by researchers from Universitat Autònoma de Barcelona. This project is the basis for two works analyzed in our review [21, 24].

Neural Network Eye Tracker (NNET) [41] is the NN-based eye tracker implementation for iPad devices, which is presented in two articles [40, 92]. EyeTab [132] is another open source codebase for tablet computers, which uses the iris contour model-based method described by Wood and Bulling [131].

Recently, the TurkerGaze project [138, 139] was made available on GitHub. This application is totally implemented in JavaScript (JS), which makes it platform-independent (with possible extension to the mobile). The library has a polished interface for calibration and verification, and comes with a small application for analyzing the gaze patterns recorded during conducted experiments. Although its proposed usage area is to enable crowdsourcing eye tracking tasks on platforms similar to Amazon Mechanical Turk, we believe it will have a larger impact on both academic works and web-based applications.

One last open source application is Camgaze [120], which is written in Python and calculates binocular gaze estimations.

2.8 Summary and Conclusions

In this chapter, we presented a review of the state-of-the-art in remote, natural light gaze trackers. Although in recent years many great works were published in the field, and the accuracy gap to reach the infrared-based trackers is closing; many open problems and unexplored approaches still remain.

Apart from the accuracy, the biggest challenges to these trackers are *a)* making the calibration less painful for the user, *b)* allowing free head movements. As we analyzed in the previous sections dedicated to these two problems, the field witnessed amazing works recently. Some open lines of work that we have identified in these areas are the following:

- **Maintaining personal calibration:** Most of the works we analyzed require some sort of calibration, be it for personal parameters for the user, for camera properties or simply for training the gaze mapping component. Although some techniques may already allow it (without stating explicitly), reusing the calibration information for the subsequent sessions of the same user is still pending extensive analysis. With such a technique, calibration before each session can be simplified or removed altogether.

- **Using calibration data from other users:** Despite being explored in a few papers [7, 144], we believe the accumulation (or collection) of training data from people other than the current user will receive more focus in the coming years. This is analogous to training classifiers or detectors in other computer vision tasks, and it will let us make better use of the large datasets that we have begun to build.

- **Other ways of collecting data:** Collecting calibration samples each time the user clicks the mouse enabled us to create very large datasets for the first time [104, 105, 144]. Especially with the advent of JavaScript-based eye trackers [138], other possibilities such as remotely crowdsourcing data collection will emerge. Larger data will eventually let us explore previously impossible ideas, a trend which is common in computer vision.

These lines of work are mostly around the topic of data collection and calibration, and they will help solve the large data needs of training for different head poses.

Most of the recent high-performing techniques [61, 66, 68, 127] are using feature-based gaze estimation, which shows the promise of this category over appearance- or model-based methods. Fig. 2.2 also shows this tendency, and the increase in feature-based methods can be observed clearly. Over the next years, we will probably see more examples of similar work with the following focus points:

- **Different features:** The PC-EC vector; pixel intensity and color; and other standard features (such as HOG, LBP) have been used so far. New feature representations that may be better suited to the problem at hand will greatly

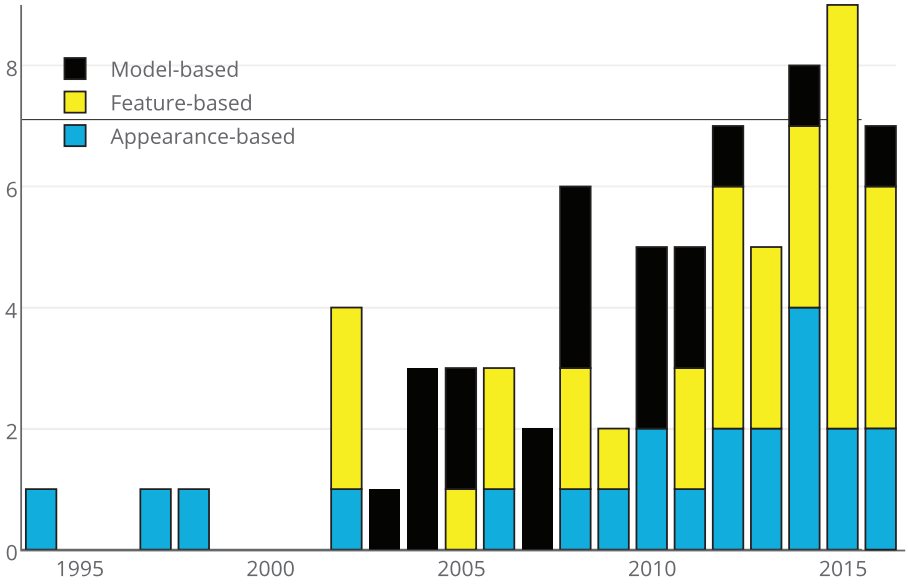


Figure 2.2 – Number of works from different categories of eye trackers according to the publication year.

improve the eye tracking accuracy. The desired characteristics of such features are: *a)* invariance to head pose, *b)* invariance to intensity changes, *c)* invariance to personal appearance differences.

- **Migrating proven ideas from other CV fields:** Use of convolutional neural networks (CNN) [144], features such as HOG, LBP, and in general the computer vision (CV) pipeline [146] are changing our approach to the gaze tracking problem. These ideas were already commonplace in other areas of CV, and we believe our community will keep transferring insights which have been proven to work for other problems.

Apart from these technical challenges and lines of work; as a society, our biggest problems are related to transparency and letting others build on our work.

Firstly, only very few of these works report their accuracy on publicly available datasets or publish the dataset they use. This is a must in other computer vision areas so that the results from techniques can be compared and verified. Moreover, standardization of the processing pipeline will immediately follow (as it depends on the training data structure) and will foster our progress.

Our second problem is that, only few works make their source code available. This prevents other researchers from *standing on the shoulders of giants*, and hinders the rate of our progress. We believe that by releasing our source code, we can create stronger ties and cooperation in the field.

In conclusion, the amount and quality of the recent work in the field is promising and signals an even faster progress in the coming years. With this *map* of the current state-of-the-art that you are holding in your hands (or gazing at through an electronic display), we hope to provide a reference point for all these amazing works we cannot wait to see.

3 Gaze Estimation Methods Based on Traditional Features

After doing an extensive review of the state–of–the–art in the previous chapter, we see that some of the methods can achieve quite low error rates (lower than 1°) [61, 68]. However, these methods mostly have been tested in experimental setups in laboratories, and are almost never made available to the general public as a final, usable research output. Without advanced programming skills and a solid background in image processing (and sometimes mathematics), it is impossible to make use of these systems.

One of our aims in this work is to build on an open source eye–tracking software [148] with all our contributions, and make the final gaze tracker available to the public at all times. With this purpose, we start with explaining our first methods for gaze estimation in this chapter. These methods have the traditional computer vision pipeline as their structure: they manipulate the input in some ways to calculate hand–crafted features, and feed these to a regression algorithm to calculate the gaze points. This is in contrast to the deep learning method we explain in the next chapter.

With these works, we explore the categories of appearance–based and feature–based gaze estimation algorithms, and make improvements on the open source baseline system.

3.1 Appearance–Based Gaze Estimation

3.1.1 Introduction

In most of the previous works on natural light gaze tracking, the focus has always been on accuracy and on the proposed method itself. The same experimental setup is used throughout the experiment, and no variations are introduced to assess the affects of the changed parameter in the setup. We believe that knowing these effects will give us, the researchers, valuable insights on how to improve the overall performance and robustness.

In this part of our work, we aim to analyze the factors that are affecting the accuracy of these gaze trackers such as head pose stability, camera placement, camera resolution. In doing so, we build an easy–to–use gaze tracking system

which improves the performance of the open source system that we take as a base. Moreover, we also bundle the videos recorded in our experiments in a publicly available dataset so that other researchers in the field can benefit from this data in their work.

Our Contributions

We build our eye-tracker on top of the open source eye–tracker Opengazer, and our contributions are aimed at making the system more robust, increasing its performance and making it easier to use.

One of the differences of our work from the appearance–based methods mentioned above, is that it is a fully automatic system. The auto facial point selection mechanism initializes the system, and the rest of the application does not require much user input. Moreover, the work on the point tracking and image normalization handles the problems of illumination change and accumulated tracking errors in longer tracker use. The training error correction improves the estimations especially for targets near the screen corners. These contributions can be seen in Figure 2.1.

3.1.2 Method

The components of the software can be seen in Figure 3.1. The original system requires at least 4 facial feature points chosen manually on subject's face and it employs a combination of optical flow (OF) and 3D head pose based estimation for tracking them. The image region containing one of the eyes is extracted and used in calibration and testing. In calibration, the subject is asked to look at several target locations on the display while image samples are taken and for each target, an average eye image is calculated to be used as input to train a Gaussian process (GP) estimator. This estimator component maps the input images to display coordinates during testing.

Our first contribution is a programmatic point selection mechanism to automate this task. Then we propose several improvements in the tracking component, and we implement image intensity normalization algorithms during and after tracking. We finish the work on the blink detector to use these detections in other components. In calibration, we propose a procedure to assess and eliminate the training error. For the gaze estimation component, we try to employ a neural network method [40]. In the following subsections, we give the details of these contributions and talk about their effects on system performance in the discussion section.

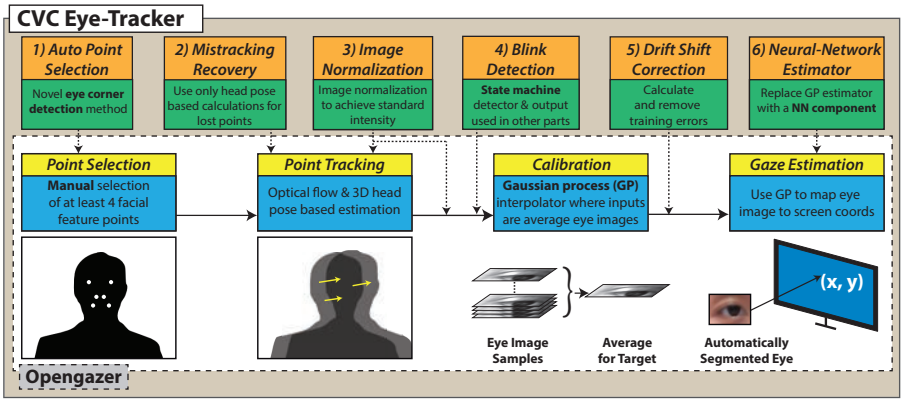
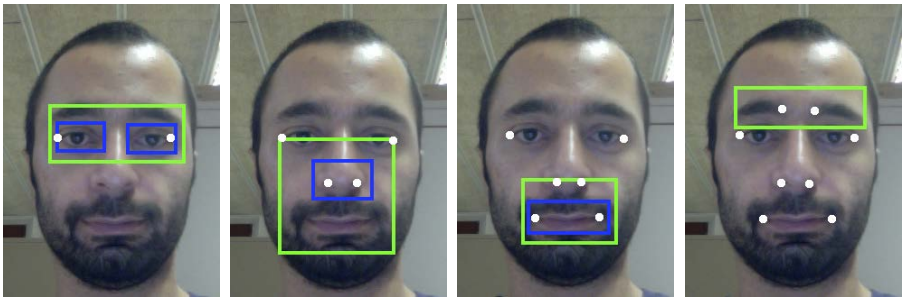


Figure 3.1 – The pipeline of the eye-tracker and our contributions on top of the base code



(a) Eye corners (b) Nose tips (c) Mouth corners (d) Eyebrow corners

Figure 3.2 – Sequence of facial feature point selection

Point Selection

Our contribution in the automation of the point selection mechanism aims at removing the errors due to operation mistakes. Moreover, it provides a standardized technique which increases the system's robustness. It employs a combination of Haar cascade detectors [14, 31], geometrical heuristics and a fast eye-corner detection technique. First, a cascade is used to detect the region containing both eyes and then the novel method detects the outer eye-corner points (Figure 3.2a). Here, the proposed method extracts all corner points inside the ROI using Harris detector, and calculates the average corner coordinates in the left and right half of the region. These two points are considered as approximate eye centers and the outer corner points are chosen on the line that passes through them. As we only search a point around the eye corner that is stable enough, we do not make more complex calculations and we simply choose the eye corner points at a predefined distance ($^1/_3$ of the distance between two centers) away from the center point approximates.

After the eye corners are selected, we search the nose in a square region below them. When the Haar cascade returns a valid detection—as in the inner rectangle in Figure 3.2b—, the two nasal points are selected at fixed locations inside this area. The algorithm continues in a similar way for the mouth and eyebrow feature points.

Point Tracking

The point tracking component of the original system uses a combination of optical flow (OF) and 3D head pose based estimation. Optical flow calculations are done between the current camera image and the previous image. This methodology results in the accumulation of small tracking errors and causes the feature points to deviate vastly from their original positions after blinking, for instance. In order to make our eye-tracker more robust to these problems, we modified the tracking component so that OF is only calculated against the initial image saved while choosing the feature points. Moreover, if we still lose track of any point, we directly use the estimate calculated using the 3D head pose and correctly tracked points' locations.

Image Normalization

During eye-tracker usage, the ambient light may change depending on the sun or other external light sources. Particularly, the computer display itself also acts as a source of frontal illumination, which contributes to a modification of the shades and shadows on the face as images of different intensities are shown in the screen. As the gaze estimation component of our eye-tracker uses intensity images for training and testing, the change in the light level is reflected in the increased error

rates of the system.

Normalization Techniques In order to tackle the varying lighting conditions, we incorporate two image normalization techniques to standardize the intensities over time [30]:

a) **Standard pixel intensity mean and variance:** In this technique, we first calculate the mean (μ^{orig}) and the standard deviation (σ^{orig}) of the original image (I^{orig}) pixels. In the next step, the scale factor (S) is calculated as:

$$S = \frac{\sigma^{\text{norm}}}{\sigma^{\text{orig}}} \tag{3.1}$$

where σ^{norm} is the desired standard deviation of intensity for the normalized image (I^{norm}). Finally, the normalized image pixels are calculated with the formula:

$$I^{\text{norm}} = S \cdot (I^{\text{orig}} - \mu^{\text{orig}}) + \mu^{\text{norm}} \tag{3.2}$$

Here, the equation first scales the image pixels to have the desired standard deviation, then shifts the mean intensity to the desired value.

b) **Standard minimum and maximum intensity:** The second method aims at normalizing the images so that the minimum and maximum intensity values are the same among all the images.

We start by calculating the minimum ($\min^{\text{orig}}$) and maximum ($\max^{\text{orig}}$) pixel intensities in the original image. Then, the scale factor is calculated as:

$$S = \frac{\max^{\text{norm}} - \min^{\text{norm}}}{\max^{\text{orig}} - \min^{\text{orig}}} \tag{3.3}$$

which is basically the ratio of pixel intensity interval between the desired normalized image and original image. Lastly, the normalized image pixels are calculated as:

$$I^{\text{norm}} = S \cdot (I^{\text{orig}} - \min^{\text{orig}}) + \min^{\text{norm}} \tag{3.4}$$

where the image pixels are mapped from range $[\min^{\text{orig}}, \max^{\text{orig}}]$ to $[\min^{\text{norm}}, \max^{\text{norm}}]$.

Variations in Usage Having these two normalization techniques at hand, we continue by incorporating them in the eye-tracker. Normalization takes into account the distribution of gray levels for a given region. In our particular context, this can be

applied in a pyramidal approach to: 1) the region containing the eye, 2) the region containing the face, or 3) the whole image. Since the statistics of each region are different, normalization is expected to provide different results depending on the region of application. In addition, normalizing in different regions has an impact in a number of system modules as explained next:

- **Eye-region normalization:** Only the extracted eye regions are used for the normalization. By applying normalization to the eye-regions we guarantee that the gaze estimation component always receives images with similar intensity distributions.
- **Face-region normalization:** Normalization parameters are derived and applied to the face region. We make use of the facial feature points' positions for a fast region segmentation. The bounding box coordinates for these points are calculated and the box is expanded by 80% horizontally and 100% vertically so that the whole face is contained. By normalizing within the face bounding box we aim at improving point tracking by removing the effects of intensity variations.
- **Whole-image normalization:** By using the whole image, we adapt the normalization to the average light conditions. However, variations in the background can affect the final result. Potentially, changes in the frontal illumination provided by the display can affect stability of the facial feature points detection.
- **Combined normalization:** Lastly, we apply the eye-region normalization on top of the face-region or whole-image normalizations. By combining both methodologies, we expect to address both the tracking problems and the problems caused by not normalized eye-region images.

Blink Detection

The blink detector is an unfinished component of Opengazer and we continue with analyzing it and making the necessary modifications to get it running. We believe that blinks have an effect on performance and by skipping them during training, we can remove the errors they introduce.

The blink detector is designed as a state machine with initial, blinking and double blinking states. The system switches between these, depending on the differences in eye images that are extracted as described in the previous section. These differences are calculated as the L2 norm between the eye images in consecutive frames. When the difference threshold for switching states is exceeded during several frames, the state is switched to the next state and a blink is detected.

We built on this structure and completed the rules for the state switching mechanism. Moreover, we added a state reset rule that resets the system to the initial state whenever the threshold criteria is not met at a certain frame.

Calibration

The original system uses all the images acquired during the calibration step. We propose a modification in the calibration part which uses our blink detector so that the images acquired during blinks are no longer included in the calibration procedure. This is crucial because these frames can alter the average eye images calculated during calibration and therefore are reflected as noise in the calibration procedure. However; as these frames are no longer available for calibration, we have to increase the time each target point is displayed on the screen in order to provide the system with enough samples during calibration.

Another improvement that we propose is the correction of calibration errors as illustrated in Figure 3.3.

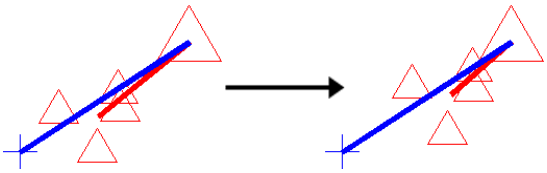


Figure 3.3 – The drift correction moves the estimates (small signs) towards the actual target (larger signs). The training error direction (blue line) and testing error direction (red line) show the correlation.

Here, red triangles on the left side correspond to a target point displayed on the screen and the corresponding gaze estimations of our system, one for each camera frame. The larger symbol denotes the actual target, whereas the smaller ones are the estimates. The shorter line connects the average estimation and the target location. Therefore, the length and direction of this line gives us the magnitude and direction of average testing error. Apart from these symbols, the longer line that starts from the target denotes the direction of the calibration error. However, it should be noted that in order to easily observe the direction, the magnitude of the calibration error is increased by a factor of 5. In this figure, we can see the correlation between the calibration error and average testing error, therefore we propose a correction method. The final effect of this technique can be seen on the right side, where the estimates are moved closer to the actual target point.

To calculate the calibration errors, we store the grayscale images which are used

to calculate the average eye images during calibration. Therefore, we save several images corresponding to different frames for each target point. After calibration is finished, the gaze estimations for these images are calculated to obtain the average gaze estimation for each target. The difference between these and the actual target locations gives the calibration error.

After the calibration errors are calculated, we continue with correcting these errors during testing. We employ two multivariate interpolators [123, 124] which receive the average gaze estimations for each target point as inputs and are trained to output the actual target x and y coordinates they belong to. The parameters that we chose for the interpolators are: approximation space dimension $ndim = 2$, Taylor order parameter $N = 6$, polynomial exactness parameter $P = 1$ and safety factor $safety = 50$. After the interpolator is trained, we use it during testing to remove the effects of calibration errors. We pass the currently calculated gaze estimate to the trained interpolators and use the x and y outputs as the corrected gaze point estimation.

Gaze Estimation

Originally, gaze estimates are calculated using the image of only one eye. We propose to use both of the extracted eye images to calculate two estimates. Then, we combine these estimations by averaging.

We also consider the case where the GP interpolator is completely substituted in order to see if other approaches can perform better in this particular setup. Neural network (NN) methods constitute a popular alternative for this purpose. There exist recent implementations of this technique [40]. In the aforementioned work, an eye-tracker using NNs to map the eye image to gaze point coordinates is implemented and is made available [41].

We incorporated the NN method in our system by making use of the Fast Artificial Neural Network (FANN) library [80] and created a similar network structure, and a similar input-output system as the original work. Our neural network had 2 levels where the first level contained 128 nodes (1 for each pixel of 16×8 eye image) and the second level contained 2 nodes (one each for x and y coordinates). We scaled the pixel intensities to the interval [0, 1] because of the chosen sigmoid activation function.

Experimental Setup

In this section, we give the details of the experimental setup we created to test the performance of our application. Variations in the setup are introduced to create separate experiments which allow us to see how the system performs in different conditions. Figure 3.4 shows how the components of the experimental setup are

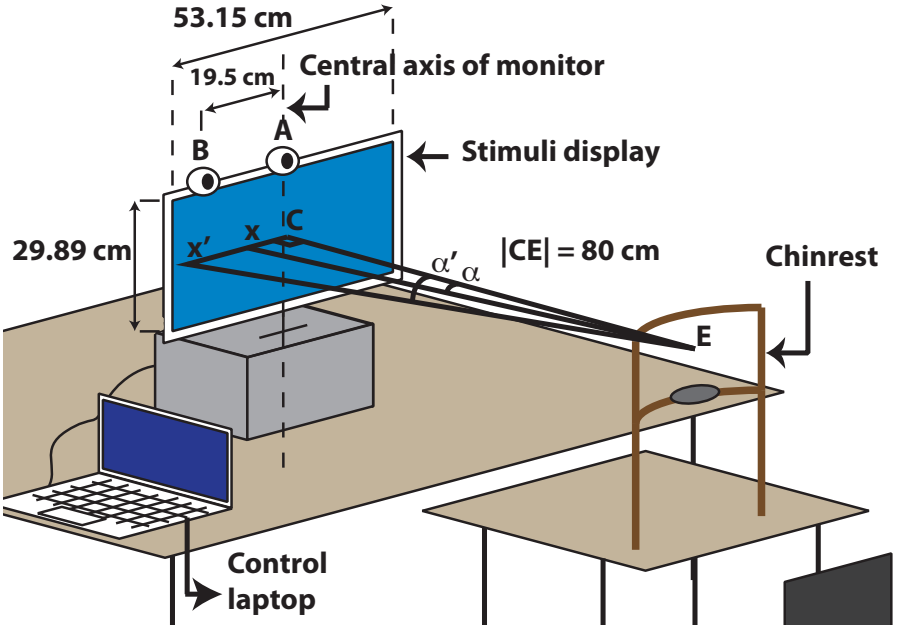


Figure 3.4 – Placement of the components in the experimental setup and the geometry involved in error calculation

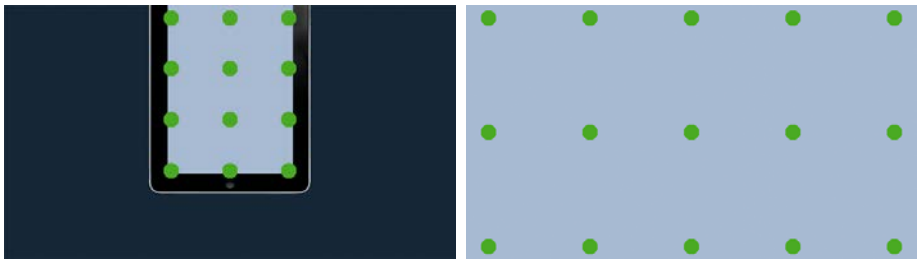
placed in the environment.

The stimuli display faces the subject and it is raised by a support which enables the subject to face the center of the display directly. The camera is placed at the top of this display at the center (A), and it has an alternative location which is 19.5 cm towards the left from the central location (B). An optional chinrest is placed at the specific distance of 80 cm away from the display, acting as a stabilizing factor for one of the experiments.

By introducing variations in this placement, we achieve several setups for several experiments which test different aspects of the system. These setups are:

Standard setup: Only the optional chinrest is removed from the setup shown in Figure 3.4. Subject's face is 80 cm away from the display. The whole screen is used to display the 15 target points one by one.

Extreme camera placement setup: This setup is similar to the previous one. The only difference is that the camera is placed at its alternative location which is



(a) iPad setup (b) Other setups

Figure 3.5 – Target positions on the display for different setups

19.5 cm shifted towards the left. The purpose of this setup is to test how the position of the camera affects the results.

Chinrest setup: A setup similar to the first one. The only difference is that the chinrest is employed. This experiment is aimed at testing the effects of head pose stability in the performance.

iPad setup: This setup is used to test the performance of our system simulating the layout of an iPad on the stimuli display. This background image contains an iPad image where the iPad screen corresponds to the active area in the experiment and is shown in a different color (see Figure 3.5a). The distance of the subject is decreased to 40 cm, in order to simulate the use-case of an actual iPad. The camera stays in the central position; and it is tilted down as seen necessary in order to center the subject's face in the camera image.

We also analyze the effect of different camera resolutions in these setups. This is done in an offline manner by resizing the original 1280×720 image to 640×480.

The error in degrees is calculated with the formula:

$$Error = |\arctan(D_{xC}/D_{EC}) - \arctan(D_{x'C}/D_{EC})|$$

where, x is the target, x' is the estimate, C is the display center and E is the face center point. The variables D_{xC}, D_{EC} and so on denote the distances between the specified points. They are converted from pixel values to cm using the dimensions and resolution of the display.

For the evaluation of normalization techniques, the videos recorded for the standard setup are processed again with one of the normalization techniques incorporated into the eye-tracker at a time.

Table 3.1 – Errors in degrees with 1280 × 720 camera resolution for all setups

	Standard		Extreme	
Version	Hor. (σ)	Ver. (σ)	Hor. (σ)	Ver. (σ)
ORIGINAL	**1.80** (0.75)	**1.46** (0.53)	**2.02** (1.11)	**1.92** (0.57)
2-EYES	1.49 (0.62)	1.42 (0.54)	1.50 (0.60)	1.66 (0.40)
TRACKING	1.59 (0.56)	1.41 (0.50)	1.71 (0.74)	1.77 (0.35)
BLINK REMOVAL	1.59 (0.52)	1.42 (0.52)	1.73 (0.74)	1.79 (0.37)
CORRECTION	**1.47** (0.54)	**1.35** (0.50)	**2.24** (2.07)	**1.95** (0.99)
NEURAL NETWORK	4.90 (1.39)	1.97 (0.67)	4.86 (1.93)	2.17 (0.44)
	Chinrest		iPad	
Version	Hor. (σ)	Ver. (σ)	Hor. (σ)	Ver. (σ)
ORIGINAL	**1.25** (0.55)	**1.53** (0.49)	**2.80** (1.17)	**2.58** (0.80)
2-EYES	1.15 (0.95)	1.40 (0.39)	2.47 (1.07)	2.53 (0.95)
TRACKING	1.18 (0.87)	1.40 (0.44)	2.02 (0.83)	2.03 (0.53)
BLINK REMOVAL	1.17 (0.90)	1.37 (0.41)	2.01 (0.84)	1.98 (0.52)
CORRECTION	**1.07** (0.85)	**1.26** (0.39)	**1.83** (0.83)	**1.86** (0.59)
NEURAL NETWORK	4.18 (1.17)	2.01 (0.51)	5.97 (0.93)	3.83 (1.55)

3.1.3 Results

In this section, we present the results which show the effects of the proposed changes on the performance. To achieve this, we reflect our changes on the original Opengazer code one by one and compare the results for all four experiments. We compare 6 different versions of the system which denote its certain phases:

a) **ORIGINAL:** Original Opengazer application + automatic point selection

b) **2-EYES:** Previous case + average estimate of 2 eyes

c) **TRACKING:** Previous case + tracking changes

d) **BLINK REMOVAL:** Previous case + excluding blinks during calibration

e) **CORRECTION:** Previous case + training error correction

f) **NEURAL NETWORK:** Previous case + neural network estimator

In all versions, the facial feature points are selected automatically by the method described in previous sections and gaze is not estimated during blinks. For each experiment, average horizontal and vertical errors for all subjects and all frames are given in degrees and the standard deviation is supplied in parentheses.

Table 3.1 shows the progressive results of our eye-tracker's performance for different versions of the system. Each result column denotes the horizontal or

Table 3.2 – Standard setup errors in degrees with 640 × 480 resolution

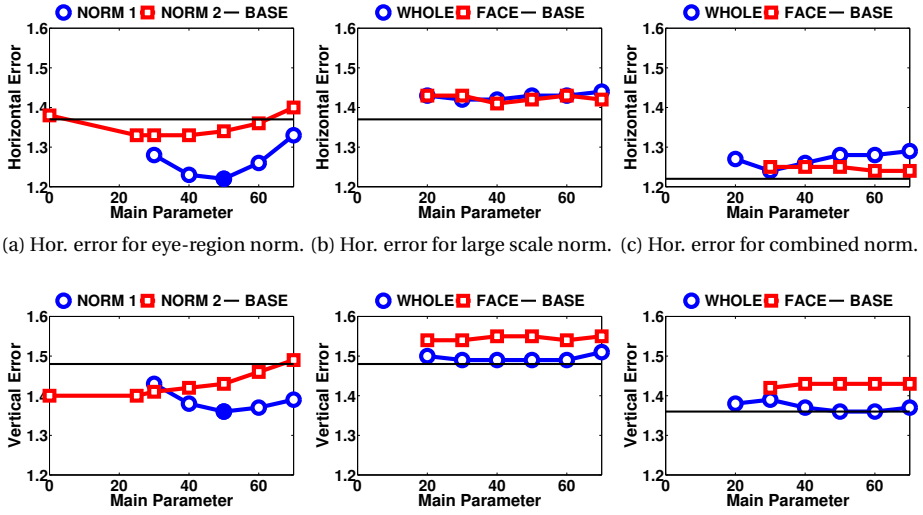
	Standard	
Version	Hor. (σ)	Ver. (σ)
ORIGINAL	**1.64** (0.66)	**1.45** (0.63)
2-EYES	1.40 (0.65)	1.36 (0.65)
TRACKING	1.54 (0.72)	1.48 (0.66)
BLINK REMOVAL	1.54 (0.68)	1.48 (0.66)
CORRECTION	**1.42** (0.72)	**1.40** (0.65)
NEURAL NETWORK	4.71 (1.29)	2.20 (0.84)

vertical errors for a different experimental setup. Moving from top to bottom in each column, the effects of our changes can be seen for a single error measure of an experimental setup. Along each row, the comparison of errors for different setups can be observed. Table 3.2 shows the performance values of the system in the standard setup with the lower resolution camera. These results can be compared to the high resolution camera's results as seen in Table 3.1 to see how the camera resolution affects the errors in the standard setup. The original application's results (ORIGINAL) and our final version's results (CORRECTION) are shown in boldface to enable fast comparison.

As for the normalization contributions, the results are grouped in Figure 3.6. Figure 3.6a and 3.6d show the results for the two techniques applied on the eye images. The black baseline shows the best results of the system without any normalization (1.37° horizontal, 1.48° vertical errors). For the **standard pixel intensity mean and variance** normalization (**NORM 1**), the standard intensity mean parameter is fixed to 127 and several values are evaluated as the standard deviation parameter (main parameter). This choice was made because when the standard deviation parameter is selected, the mean parameter does not affect the gaze estimations unless it results in the trimming of pixel intensities (mapping to intensities outside the range $[0,255]$). In the case of **standard minimum and maximum intensity** normalization (**NORM 2**), the minimum intensity parameter is considered the main parameter and the maximum intensity is set to: $\max^{\text{norm}} = 255 - \min^{\text{norm}}$.

In Figure 3.6b and 3.6e, the second set of normalization results are shown. Here, the better performing **NORM 1** technique is applied to the whole camera image (**WHOLE**) or the face-region (**FACE**) and the results are given.

Lastly, Figure 3.6c and 3.6f show the results when the eye-region normalization is combined with whole-image or face-region normalizations. As the eye-region normalization is independent of the others, the parameters for this step are fixed



(a) Hor. error for eye-region norm. (b) Hor. error for large scale norm. (c) Hor. error for combined norm.

(d) Ver. error for eye-region norm. (e) Ver. error for large scale norm. (f) Ver. error for eye-region norm.

Figure 3.6 – Errors in degrees for eye image normalization techniques. **(a) and (d)** show errors for eye-region normalization. In **(b) and (e)**, the errors for only face-region and whole-image normalization methods are shown. These methods apply the first normalization technique in the corresponding image areas. Lastly, **(c) and (f)** show the results where the eye-region normalization is applied on top of either the whole-image or the face-region normalization. The black straight lines show the baseline accuracy for comparison. In **(a), (b), (d) and (e)**, this corresponds to the error rate without any normalization. In **(c) and (f)**, it denotes the best results so far, which belong to the eye-region normalization using technique 1 (filled data points in (a) and (d)).

to the best performing values ($\mu^{\text{norm}} = 127$, $\sigma^{\text{norm}} = 50$) and the black baseline shows the best results achieved with only eye normalization (1.22° horizontal, 1.36° vertical errors).

3.1.4 Discussion

Considering the **1.47**° horizontal and **1.35**° vertical errors of the final system in the standard experimental setup, we conclude that we have improved the original system by **18%** horizontally and **8%** vertically. As seen in Table 3.2, the performance

difference in the same experiment done with VGA cameras (**13%** horizontally, **3%** vertically) is comparably lower than the first case, which shows us that our contributions in this work exhibit more robust performance with the increased image quality. From another aspect, it means that better cameras will favor the methods we proposed in terms of robustness.

One interesting aspect of these results is that with the increased camera resolution, the original application shows a worse performance. We believe this is caused by the optical flow algorithm used in the tracking component. The increased detail in the images affect the tracking and the position of the tracked point may vary more compared to the lower resolution image. This, combined with the accumulated tracking error of the original application, result in a higher error rate. However, it is seen that the final version of our eye-tracker (**CORRECTION**) recovers most of this error.

From the extreme camera placement setup results seen in Table 3.1, we see that shifting the camera from the top center of the display decreased the performance by **52%** horizontally and **44%** vertically. Here, the performance loss is mainly caused by the point tracking component. From such a camera angle, the farther eye corner point may be positioned on the face boundary, making it hard to detect and track. When we compare the errors before and after the error correction is applied (**BLINK REMOVAL** and **CORRECTION**), we see that this change introduced a great amount of error for this case. We can say that the unreliable tracking also hinders the error correction component, because the correction relies on the calibration being as good as possible. In order to tackle these problems, a 3D model based face tracking algorithm may be employed.

In the 3rd experimental setup, we show that the chinrest improves the performance by **27%** horizontally compared to the standard setup. This setup proves to be more reliable for experimental purposes.

The results for the iPad setup may be deceiving, because here actually the errors in pixels are lower; however, as the distance of the subject is used in the calculation of errors in degrees, the angular errors are higher. Each **1**° error in other setups corresponds to around twice as many pixels on the screen compared to a **1**° error in the iPad setup. Using this rule of thumb, we can see that the iPad case results in lower error rate in pixels compared to even the chinrest setup.

One of the major problems with the original system lies in the tracking component. As tracking is handled by means of optical flow (OF) calculations between subsequent frames, the tracking errors are accumulated and the facial feature points end up far away from their original locations. To tackle this problem, we proposed to change this calculation to compare the last camera frame with the initial frame which was saved during facial feature point selection. Comparing the **2-EYES** and **TRACKING** results in Table 3.1, we see that the tracking changes increased the

system's accuracy in the iPad setup. However, in the first two setups we have just the opposite results. This is probably because when using both of the eyes for gaze estimation, the tracking problems with the second eye have a larger effect on the averaged gaze estimation.

We observe that excluding the blink frames from the calibration process (application version labeled **BLINK REMOVAL**) does not have a perceivable effect on the performance. We argue that the averaging step in the calibration procedure already takes care of the outlier images. The neural network estimator does not provide an improvement over the Gaussian process estimator and performs similar to its reported accuracy (4.42°). We believe this is due to eye images extracted by our system. Currently the feature point selection and tracking mechanism allows small shifts in point locations and therefore the extracted eye images vary among samples. The GP estimator resolves this issue during the image averaging step; however, the NN estimator may have problems when the images vary a little in the testing phase. In order to resolve this problem, a detection algorithm with a higher accuracy may be used to better estimate the eye locations.

Analyzing the eye image normalization results as seen in Figure 3.6a and 3.6d, we see that both approaches improve the results. For the first normalization technique, the parameter value giving the best results is $\sigma^{\text{norm}} = 50$, which decreases the errors by **11%** horizontally and **8%** vertically. We can say that the eye image normalization does just what the Gaussian process estimator needs and helps compare eye images from different time periods in a more accurate way. The results for the second normalization method lag behind especially in the horizontal direction.

Figure 3.6b and 3.6e show the results for intensity normalization in the large scale, either applied to the whole camera image or around the face region. Here, we see that large scale normalization applied on top of eye image normalization does not improve the results at all. Our expectations for the face normalization to perform better than whole image normalization are not verified, either. We observe that the face normalization performs especially worse in vertical direction.

As it can be seen in Figure 3.6c and 3.6f, combining the two normalization methodologies do not increase the system's accuracy, either. From these results, we can conclude that face-region or whole-image normalization cause the tracking component perform worse, and thus decrease the performance.

3.1.5 Conclusion

Our contribution provides significant improvement in a number of modules of the baseline system. The automatic point selection technique enabled us create an easy to use application, removing errors caused by wrong operation. The experiments showed that the final system is more reliable in a variety of scenarios. The blink

detection component is mostly aimed at preparing the eye-tracker to real world scenarios where the incorrect estimations during blinks should be separated from meaningful estimates. The proposed error correction algorithm helped the system better estimate gazes around the borders of the monitor. Lastly, the eye image normalization technique improved the performance of the system and made the system more robust to lighting conditions slightly changing in time. The final code can be downloaded from the project page [22].

Apart from these experimental performance assessments, this work resulted in an additional valuable output, which is the **publicly available dataset** [23] that consists of the videos recorded during the experiments. This dataset contains videos for 12 subjects and 4 different experiment cases, for a total of 48 videos. The videos are annotated with the timestamps where specific phases such as anchor point initialization, calibration and testing take place. Moreover, the locations of the displayed on–screen gaze targets are also available for each video frame.

3.2 A Quick Feature–Based Gaze Estimation

3.2.1 Introduction

In our review in Chapter 2, we talked about a categorization scheme for gaze trackers, which groups them into three main categories: appearance–based, feature–based and model–based methods. Later on, in the previous section, we described our first proposal for an appearance–based gaze estimation method, which provided promising results. However, our experiments in that work showed us that with these types of methods, small head movements may lead to the invalidation of gaze tracker calibration, without an easy way to recover. On the other hand, model–based methods are mostly explored; with previous works analyzing 2D iris boundary [121, 122], 3D eyeball models [1, 134], as well as full face models [15, 50] for gaze tracking. This doesn't leave much creativity for proposing new types of models, but only improvements of previously proposed models (i.e. better facial structure estimation, improved iris boundary detection, etc.). We believe that the remaining category of feature–based methods still has a lot of potential for exploitation.

In this section, we propose a feature–based gaze estimation method, which is based on accurate segmentation of the iris region. Here, the iris segmentation is converted into a feature representation, which encodes the segmented pixel positions and numbers into a compact representation.

We start our contributions by a robust method for iris localization, and continue with an improvement on anchor point tracking with the help of eye corner patch extraction during initialization. Then, we explain our proposal for iris segmentation,

which is later used for extracting two types of features. Finally, we use Gaussian Process regression for mapping these features to gaze estimations.

3.2.2 Methodology

Iris Detection

As our feature proposals are based on the iris segmentation, one of the key components of this work is accurate detection and segmentation of the iris region.

Once we have the eye regions extracted using the same pipeline explained in Chapter 3, we test all the possible locations of the iris inside the 128×64 image. An illustration of the overall pipeline can be seen in Figure 3.7

As the first step, we apply an ellipsoidal mask to filter out the corners of the eye images, which may often include shadows caused by the nose. Using this mask, we fill the corner areas with an average gray color, suppressing the likelihood of finding an iris candidate in these areas.

We prepare iris templates of several sizes (with a radius between 13 and 31 pixels) to be used in calculating the likelihood of a region containing the iris. These templates consist of a black circle with the given radius, surrounded by small white margin filling the enclosing square block. A similar representation was previously used for iris localization in the literature [92], especially due to its fast calculation.

The iris templates of varying sizes are used to calculate the cross-relation between the template and the eye image. At this step, we apply all the prior information we have about how a good iris candidate should be: it is more likely to find it closer to the center of the image (assuming we extracted the eye regions correctly). Moreover, if we see similarly scored iris candidates of different sizes, the larger one is more likely to be the real iris detection, and may even be surrounding completely the smaller candidate.

To apply these pieces of prior information, we multiply the cross-correlation scores with the following:

Prioritizing the center: We apply a Gaussian (with standard deviation of 200) centered around the image center to all cross-correlation result matrices (for all template sizes). This causes the candidates at the corners of the eye images to have around 6–7% less likelihood than the candidate at the center of the image.

Proritizing larger sizes: We calculate prioritization factors for each iris template radius, of the following form: $1 + (i * i)/300$. Here, the i corresponds to the index of the iris template radius, and is in the range of $[0, 19]$. This multiplication causes the largest possible iris candidate (with $i = 19$ and $radius = 31$)

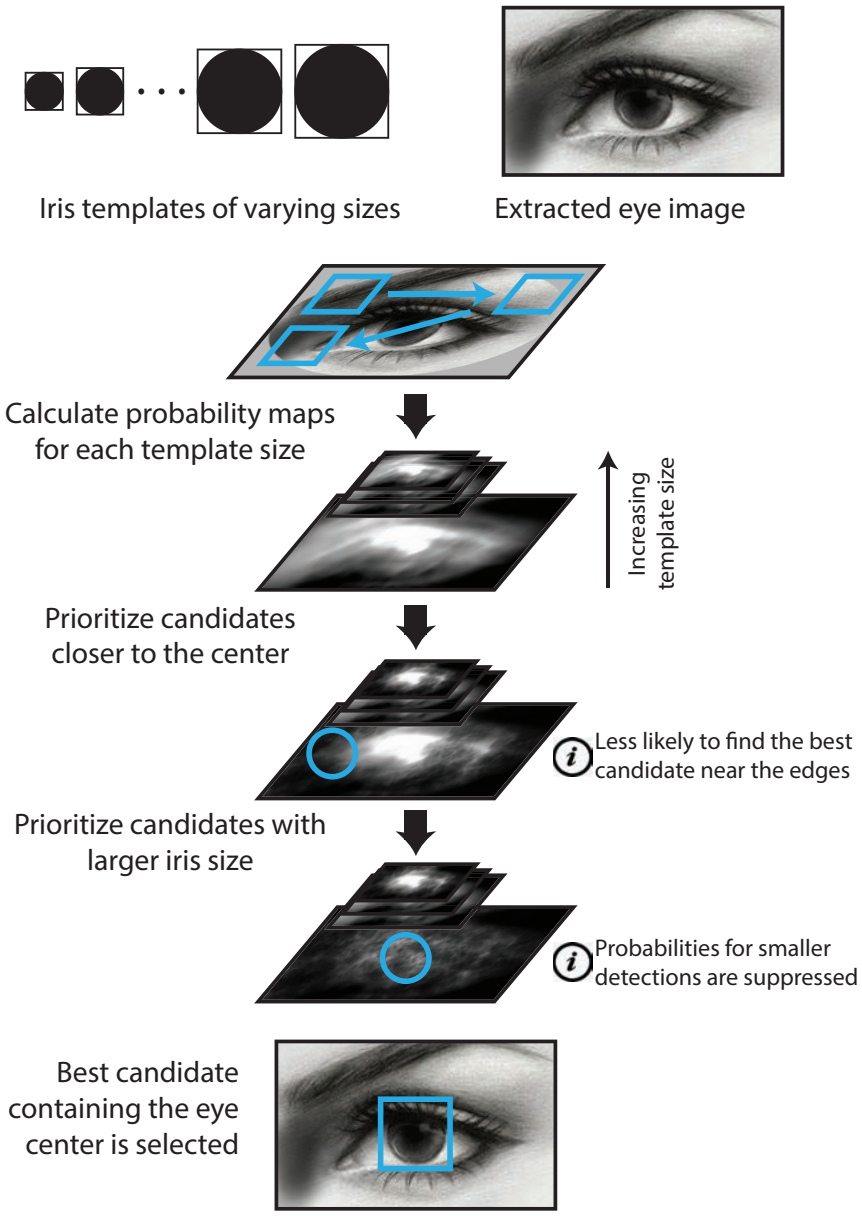


Figure 3.7 – The pipeline of the iris detection complonent

to have around 120% more likelihood than the smallest candidate.

After calculating the cross-correlations and applying the priotizations based on our domain information, we end up with all possible iris location candidates sorted by the likelihood. To increase the robustness of this method to shadows, etc., we apply one last level of filtering at this step. We use a fast eye center localization method [110] to make sure that the selected candidate contains the eye center. This method was previously used for gaze estimation [131], and shown to work well with closer images of the eye region as seen in the tablet use case. However, in the case of remote gaze tracking with a webcam, we found that the images were too low quality and the estimated center may exhibit jumps from time to time. Here, we only use it to filter out the iris candidates that our system mistakenly ranked high up.

Anchor Point Initialization and Tracking

As in our previous proposal, before the calibration, our system chooses several anchor points on the subject's face and tracks them over time.

Out of the 8 points we choose and track, the outer eye corners have a bigger impact in the overall accuracy. This is because they are directly connected to the robust extraction of eye region images which in turn are used in the feature extraction and gaze estimation modules. In the work described in this chapter, we use a special method for initializing these two points, as their accurate localization is also crucial for the newly proposed features.

Once we detect the rectangular region containing both eyes using two Viola–Jones detectors [89, 119] (one for full face, another one for two–eye region), we use the right and left halves as initial bounding boxes for the eyes. We apply the iris detection algorithm described above to locate both irises, and use the center of the iris bounding boxes as eye center approximations. The line passing through these two points is calculated, and the eye corners are selected at a predefined distance (30% of the distance between the eye centers) away from the centers.

After calculating the initial positions of outer eye corners, two small patches are extracted around these locations. When the optical flow component is done tracking the anchor points between subsequent frames, we use the previously extracted patches to finetune the position of the outer eye corners.

Feature Extraction

In the previous components of our system, we robustly detect the iris bounding box, and make sure the anchor point tracking and eye image extraction are running in a stable way. To use these information for gaze estimation, here we propose our novel feature descriptors.

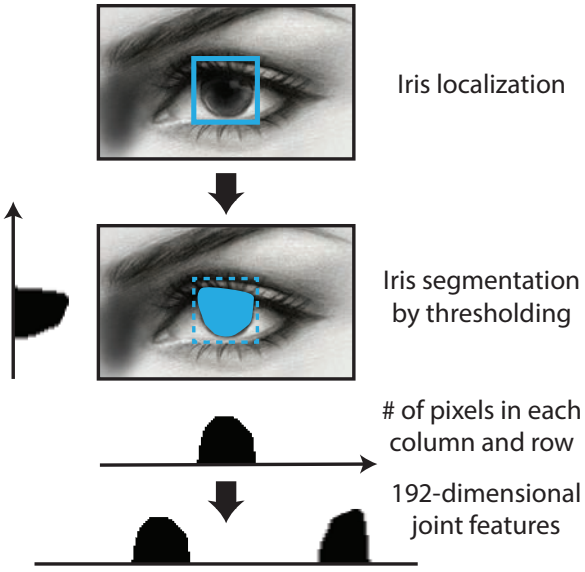


Figure 3.8 – Feature extraction steps

The first step in calculating our features is segmenting the iris region inside the iris bounding box. We apply an adaptive thresholding using Otsu's method to mark the pixels belonging to the iris (the blue blob).

Traditionally, methods use the calculated iris segmentation in several ways. One option is using it for fitting a model such as an ellipse for the iris boundary, or even more complex models where this ellipse is then used for inferring the 3D eyeball properties. Another option is calculating the iris center from the segmentation, which can later be used in gaze estimation through a variety of ways; such as calculating the pupil center-eye corner (PC-EC) vector or calculating the iris normal vector in 3D.

In our method, we aim for creating a novel feature representation which can encode this information in a meaningful way.

As seen in the figure, feature extraction takes as input the iris segmentation and creates two types of features: *vertical features* are calculated by a projection of segmented iris pixels into the vertical axis, and *horizontal features* are calculated by projection to the horizontal axis. Here, projection operation sums the number of pixels in each row (for vertical features) and in each column (for horizontal features).

The intuition behind these features is that: when we use iris segmentation (or detected iris boundary pixels) to calculate features that summarize the information too much (such as iris center location), we lose a lot of relevant information. If we encode the distribution of positions of all these pixels in our feature set, our regression component can make use of it to better estimate the gaze direction.

When we compare this feature set with our first proposal from Chapter 3, we see that in this new representation we are only keeping the relevant information. In the previous proposal, all image pixels were considered by the Gaussian Process regression component. In case of a non-homogeneous change in the lighting conditions, even the skin pixels that are irrelevant for the gaze could affect the results.

As the eye images have 128×64 pixels of resolution in this work, the horizontal and vertical features have 128 and 64 dimensions, consecutively. By joining these two vectors, we come up with the final 192 dimensional feature set.

Gaze Estimation

Once we have the features calculated, the remaining task is mapping these to screen coordinates of the gazed point. As in our previous work, we use a Gaussian Process estimator as the mapping component.

During calibration, we store the average histogram features for each calibration target on the screen. After building the training data this way, we need to decide on a kernel function for the Gaussian Process, which calculates the similarity between two samples.

Our features are basically histograms, where segmented pixels are placed into bins corresponding to their location. Thus, the logical option for the kernel function is to turn to standard histogram comparison methods. Some options that we evaluated at this step were:

Correlation, which is defined as:

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H_1})(H_2(I) - \bar{H_2})}{\sqrt{\sum_I (H_1(I) - \bar{H_1})^2 \sum_I (H_2(I) - \bar{H_2})^2}} \tag{3.5}$$

where $\bar{H_k}$ is the mean value of a given histogram.

Intersection kernel with the formula below:

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I)) \tag{3.6}$$

Lastly, the **squared exponential kernel** (also known as Radial Basis Function kernel or the Gaussian kernel) which we also used in our work described in Chap-

ter 3:

$$d(H_1, H_2) = \sigma^2 \exp\left(-\frac{(SSD(H_1, H_2))^2}{2\ell^2}\right) \tag{3.7}$$

where $SSD(H_1, H_2)$ is the sum of squared differences between the two histograms' bins and is calculated as:

$$SSD(H_1, H_2) = \sum_I (H_1(I) - H_2(I))^2 \tag{3.8}$$

Through an empirical analysis, we choose to use the squared exponential kernel and set its two parameters $\sigma = 80$ and $\ell = 125$.

Experimental Setup

In order to test the effects of our contributions on the eye-tracker performance, we designed an experimental setup and recorded 12 videos from 6 subjects. Next, we evaluated the performance of the appearance–based system described in the previous section and the newly introduced feature–based approach, calculating the errors for each video and for two versions of the system.

The experiments were carried out on a laptop computer with a 13" monitor. The subjects were asked to sit $60cm$ away from the monitor (compared to $80cm$ in the previous work), facing its center point. We used the same 15 point calibration and testing scheme, and for each subject these steps were repeated twice and in total 12 videos were recorded. The target point position shown on the display is assumed to be the ground truth and the error is calculated as the horizontal and vertical differences between the ground truth and the gaze estimation coordinates. These values (in pixels) are then converted to errors in degrees using the geometry as defined in the previous work.

3.2.3 Results

Table 3.3 shows the results of the feature–based gaze estimation next to the results of the appearance–based method on the newly collected data.

Figure 3.9 shows the error in degrees obtained from the dataset of 12 videos. The bars on the left show the results for the base system [24], where the average errors are 2.35° (horizontally) and 1.82° (vertically). The bars on the right show the results of histogram feature based gaze estimation, with average errors 1.54° (horizontally) and 1.61° (vertically). As observed, the proposed method decreases error in the horizontal direction by 34% and in the vertical direction by 12%.

To convert these numbers into pixel units and assess the significance on a

Table 3.3 – Comparison of gaze estimation performance between the proposed feature–based method vs. the appearance–based method from the previous chapter.

	Feature Based		**Appearance-Based**	
Experiment	Horizontal	Vertical	Horizontal	Vertical
# 01	1.13	1.18	1.81	1.39
# 02	1.36	1.39	1.29	0.92
# 03	0.80	1.31	1.58	1.58
# 04	1.07	1.18	2.21	1.42
# 05	3.23	1.97	4.27	3.00
# 06	1.66	1.38	3.06	2.59
# 07	0.97	1.55	1.47	1.25
# 08	0.99	1.63	1.54	1.37
# 09	1.89	1.27	1.70	1.31
# 10	1.38	1.32	0.71	0.88
# 11	1.51	2.31	5.25	3.74
# 12	2.49	2.78	3.28	2.45
Mean (Std)	**1.54** (0.78)	**1.61** (0.76)	**2.35** (1.35)	**1.82** (0.90)

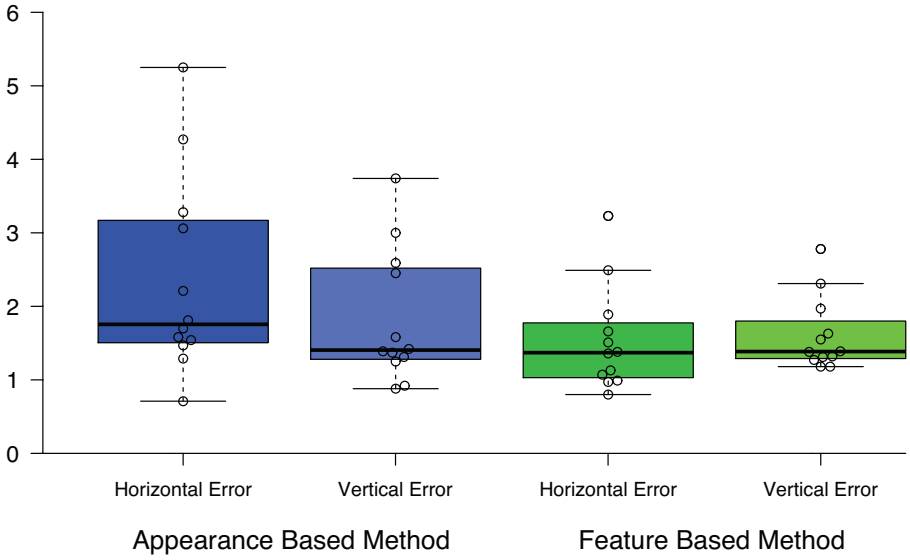


Figure 3.9 – Horizontal and vertical errors in degrees for the appearance based and feature based methods. Center lines show the medians; box limits indicate the 25th and 75th percentiles, whiskers extend 1.5 times the interquartile range from the 25th and 75th percentiles; data points are plotted as open circles.

monitor of 1280×800 resolution, the errors in degrees can be multiplied by a conversion factor of $47 \frac{pixels}{degrees}$.

3.2.4 Conclusion

In this work, we proposed a novel algorithm for gaze estimation, which brings together the advantages of appearance based and model based estimation algorithms to create a hybrid approach. As in appearance based techniques, our algorithm is as direct as possible, neither requiring personalized 2D/3D models nor using too much computational power. It eliminates the effects of change in appearance (lighting conditions, small instabilities, etc.) just like model based algorithms, and the proposed features can be invariant to small changes in head pose without much effort. Therefore it will be possible to incorporate head pose correction algorithms as a future work. Traditionally, this has been the main problem with appearance based models (especially with classical neural networks, etc.) and we believe our approach is the first step towards solving it. Our final results show significant improvement compared to previously used appearance based estimation method.

The rest of the described work has the aim of making the eye tracker easier to use and more robust. We believe these features will be the deciding factor for the application to be used outside research laboratories and to be included in real life scenarios.

4 Head Pose Invariant Gaze Estimation

4.1 Introduction

In Chapter 3 we investigated the factors that affect the performance of a visible light gaze tracker. We concluded that extreme camera placement, which is another way of saying extreme head pose, deteriorates the performance significantly. Moreover, personal differences add a significant variance (of around 0.50° in both horizontal and vertical axes) and cause the gaze estimation to be unreliable for some people. In our personal experience, we have seen that the calibration of these trackers do not stay valid for a long amount of time (longer than 5 minutes) due to factors such as head pose changes, illumination changes, eyelid openness, which accumulate over time and manifest themselves in the final error rate.

The feature based gaze estimation method described in the second part of Chapter 3 also has its share of disadvantages. For example, eyelid openness can change during tracker usage, making larger or smaller portions of the iris to be occluded. This in turn causes our gaze estimator to wrongly conclude that the user is looking downwards, in which case a larger part of the iris also becomes occluded. The proposed feature set is not affected by small & gradual changes in lighting conditions, but in an uncontrolled environment where the user may have different facial structure, the shadows around the nose and the eye cavity may hinder the detection of anchor points and the iris localization.

All of these problems actually point in the same direction: we are lacking data when we are recording videos for our experiments with limited time and thus, limited number of subjects and setups. In other areas of computer vision, the dataset sizes grew over years and enabled the application of more complex and better performing algorithms. For example, in object detection, the reference dataset size grew from 60000 images of 32×32 images in 2009 [58] to millions of larger images in ImageNet [19] after a few years.

As we explained in our review in Chapter 2, in recent years the community of visible light gaze tracking also built several large scale datasets [42, 145] that are publicly available. Our aim in the work in this chapter is to leverage the MPIIGaze dataset [145] to address the problems we listed above.

We start by introducing the face tracker that is used track the subject's face in a

more reliable way compared to our previous works. This is crucial for uncontrolled environments where the subject may move freely during tracker usage. Once we have the initial estimations of the anchor points included in the face tracker model, we customize a generic 3D model with personal parameters learned during calibration (e.g. distance between two eyes, nose size) which is later on used to estimate the head pose in a more accurate way. With this information, we proceed to extract the eye images using the 3D geometry involved, and use it for gaze estimation. For estimating the gaze, we train a convolutional neural network (CNN) that can learn better from large datasets.

4.2 Methodology

4.2.1 Face Aligment

As our aim in this chapter is building a gaze tracker that is invariant to head poses, we start with estimating the face pose of the subject that is using the tracker. In order to have the tracker run in realtime, we need a fast and still accurate face alignment algorithms to detect the positions of facial landmarks. We take as base the work of Kazemi and Sullivan [53], who developed a method that can do this detection for an image in **1 millisecond**.

We adapt the implementation of the aforementioned alrogithm from the **dlib C++ library** [55, 56] for our purposes. The algorithm takes as input the region of interest (ROI) that contains the face to be analyzed so that the calculations are faster. For this, we first use the frontal face detector from dlib to get the bounding box of the face. The image and this bounding box are then fed into the face alignment component to get the facial landmark positions. The output contains the positions of 68 landmarks (compared to the 194 landmarks in the original work) as seen in Figure 4.1. From these points, the ones lying on the eyebrows and the mouth cannot be assumed to be stable during gaze tracker usage; because smiling, talking or expressions such as surprise would cause large movements. Moreover, the points lying around the chin are not very helpful in detecting the head pose; because after head rotations the newly detected chin points are not exactly the same points in 3D. Therefore, we also discard these points and keep only a small subset of the calculated points.

Figure 4.3 shows the landmarks that we use for our purposes. These are: *a)* sellion, *b)* right eye outer corner, *c)* left eye outer corner, *d)* nose tip, *e)* menton (chin).

In our experiments, we also tested an extended face model that included the right and left sides of the face (next to the ears) and the stomion (mount center); however, this model proved to be less stable as these points are not always reliably

Figure 4.1 – Example facial landmarks as detected by Kazemi and Sullivan's method

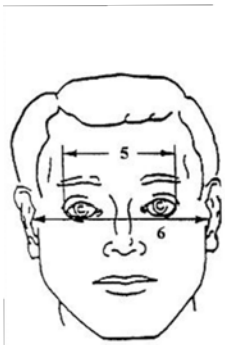
tracked.

4.2.2 Face pose estimation

With the initial localization of these five landmarks, we have a good estimate of subject's face pose. However, using a generic 3D face model to calculate the face pose leads to estimation errors. Moreover, if one decides to project a 3D point from the face model onto the image, the projected point is further away from the detected landmark position. This especially hinders our eye image extraction algorithm which needs to project several points from the 3D model onto the image for accurate image extraction.

To tackle this problem, we adapt a generic face model to the subject's face during the calibration. We take as a base the measurements of U.S. Dept. of Defense related to head anthropometry [85]. In this document, we can find detailed results about how various measures related to human body vary among the population. Of these, we focus on those focusing on facial measurements, of which an extract can be seen in Figure 4.2.

On Figure 4.3, the personal variations in facial features that we account for are also shown. In our model, the sellion is the base point (i.e. positioned at the origin).



5 **Biocular breadth.** The distance from the outer corners of the eyes (right and left ectocanthi).

	Sample		Percentiles 1st	5th	50th	95th	99th
A	Men	cm	11.0	11.3	12.2	13.1	13.6
		(in)	(4.3)	(4.5)	(4.8)	(5.2)	(5.4)
B	Women	cm	10.8	11.1	11.6	12.9	13.3
		(in)	(4.3)	(4.4)	(4.3)	(5.1)	(5.3)

6 **Bitragion breadth.** The breadth of the head from the right tragion to the left. (Tragion is the cartilaginous notch at the front of the ear).

	Sample		Percentiles 1st	5th	50th	95th	99th
A	Men	cm	13.1	13.5	14.5	15.5	15.9
		(in)	(5.2)	(5.3)	(5.7)	(6.1)	(6.3)
B	Women	cm	12.5	12.8	13.3	14.3	15.0
		(in)	(4.3)	(5.4)	(5.4)	(5.7)	(5.9)

Figure 4.2 – Extract from U.S. Dept. of Defense report on designing by taking into account anthropomorphic variations.

The positions of the rest of the facial feature points are defined relative to this point, and taking into account the personal parameters. In the model, the X axis points towards the camera, and the Z axis points upwards. The final calculations of their positions are:

Outer eye corners: $(-20 * P_{eyedepth}, \pm 59.5 * P_{eyeseparation}, 5)$

Nose tip: $(22 * P_{nosedepth}, 0, -48 * P_{noselength})$

Menton: $(0, 0, -117.5 * P_{mentonlength})$

Here, all the constants are the initial values of the facial measures such as eye depth, eye separation, etc., and they are set to the corresponding average value for the whole population (average of measures for men and women). The parameters such as $P_{eyeseparation}$ are initially set to 1.0, and the value is modified during the calibration to fit subject's face better.

During facial model calibration, we collect samples that are calculated by the face alignment component, and add a new sample to our training set whenever we detect a previously unseen head pose. This way, we have a maximum of 40 samples covering a wide range of head poses. Using this data, we apply coordinate descent to finetune the personal parameters.

In each iteration of the coordinate descent, we iterate over the personal parameters and calculate the update for the parameter. Here, we approximate the partial gradient at the current configuration relative to the updated parameter. For each

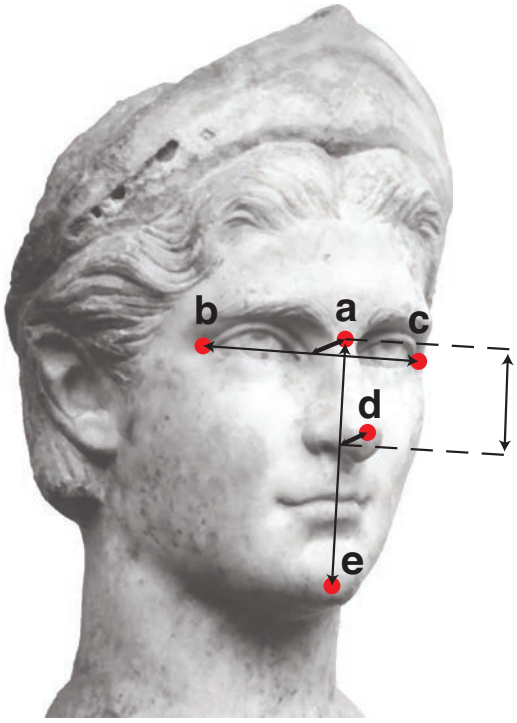


Figure 4.3 – Geometry of facial feature points and the personal parameters. **a)** Sellion, **b–c)** eye corners, **d)** nose tip, **e)** menton. The five personal parameters to calibrate are **1)** eye–sellion depth, **2)** binocular breadth (eye separation), **3)** nose depth, **4)** nose–sellion distance, **5)** menton–sellion distance.

Figure 4.4 – Eye extraction taking into account the perspective

parameter, we test how a small modification would change the projection errors on the training data. Using this as the aproximation of the gradient, we take a small step at the indicated direction. After 1000 iterations, the step size is reduced from 10^{-5} to 10^{-6}.

Once the personal parameters are calibrated, we calculate the rectangles containing the eyes in the 3D model, and project their corners onto the latest camera image. This way, we can take into account the perspective while extracting the eye images, as seen in Figure 4.4.

4.2.3 Gaze estimation

After tracking the face reliably over time, and extracting the rectangular eye images, we are left with the task of estimating the gaze with these information. In the original work [144], the authors use a well-known convolutional neural network (CNN) architecture [59] to handle this task.

CNNs have several advantages that have made them the go–to option for handling large–scale data in the recent years:

- As the weights in convolutional layers are shared for different input locations, the model is less complex and the detected features are shift invariant (i.e. the same image feature learned by a convolution can be detected anywhere in the input). Moreover, this makes training the network faster.
- Deeper architectures enable building a cascade of features. For example, a deep CNN trained to detect objects may detect parts of the object in the first layers, and the last layers build upon these detections. This way, there is no need to manually engineer the features extracted from the images.
- Available machine learning libraries such as TensorFlow [6], Theano [109], Keras [16], etc. make it easier to harness the power of GPUs and process huge amounts of data in a shorter time.

With these in mind, we also see CNNs as the best option to handle the gaze estimation task on the MPIIGaze dataset which contains > 400k samples in total. We analyzed many different architectures with the following characteristics:

- The simplest CNNs with several convolution–pooling layers, topped with a fully connected (i.e. dense) layer for finishing.
- Deeper architectures that contain many layers of convolutions.
- Models including novel ideas such as batch normalization [49], inception module [106], fire module [45], dropout [98]

Among these works, the fire modules are quite interesting as they enable more compact networks with same performance as larger networks. In this architecture, the 1×1 convolutions are used to compress (or squeeze) the input volumes before feeding them into the next convolution (which *expand* back the squeezed volumes). This way, the network has less parameters to train while still maintaining its modelling complexity. With an architecture making use of these **fire modules** (i.e. *squeeze* convolutions followed by two separate *expand* convolutions, which are later concatenated), the authors maintain AlexNet [59] level accuracy with 50× fewer parameters.

After our initial analysis, we evaluated the following options to choose the final architecture:

a) **Baseline (multi-modal input):** The baseline architecture as used in the original work [144].

b) **Deep network (VGG-like):** Deep network architecture with 5 levels of convolutional layers. This is inspired by the VGG network [94].

c) **Inception modules:** Simplified version of the original network introducing the inception modules [106]. We left only 5 inception modules in this architecture.

d) **Fire modules v1:** A modified version of the SqueezeNet architecture, making use of its fire modules [45]. The architecture can be seen in detail in Figure 4.5.

e) **Fire modules v1 (L1 reg.): v1** network, with L1 regularization in the final dense layers with a regularization factor of 0.01.

f) **Fire modules v1 (L2 reg.): v1** network, with L2 regularization in the final dense layers with a regularization factor of 0.005.

g) **Fire modules v1 (m.m. input): v1** network, with multi–modal input as described in the baseline system. The head pose angles (pitch & yaw) are concatenated to the dense layer output just before gaze estimation.

h) **Fire modules v2: v1** network, with a duplicate of the 2nd fire module to increase model depth.

i) **Fire modules v3: v1** network, with 25% more filters in each layer of fire modules (i.e. 20–80–80 instead of 16–64–64).

With the comparison of all these, we can evaluate the performances of the alternative networks against the baseline. The deep architectures have the objective of seeing whether deeper networks are better suited for this problem or not. In the variants based on fire modules, we try to see if regularization, more model complexity or late fusion of head pose information are improving the performance.

Figure 4.5 shows the final architecture employed in our experiments (base **v1** architecture). After an initial 3×3 convolution and maxpooling layers, 2 fire modules are used. In each module, first we *squeeze* the input volume with 1×1 convolutions, and then separately apply two convolutions with different sizes (1×1 and 3×3). These are then concatenated before being passed to the next layer.

After the fire modules, we apply another maxpooling to reduce the dimensionality of the output volume, and employ a dropout layer to reduce overfitting to training data. We finish by flattening the output volume and applying two dense layers to come up with the X and Y estimations. Unlike the baseline system, our experiments didn't show any performance improvements with late fusion of head pose information, and we removed it from the final architecture.

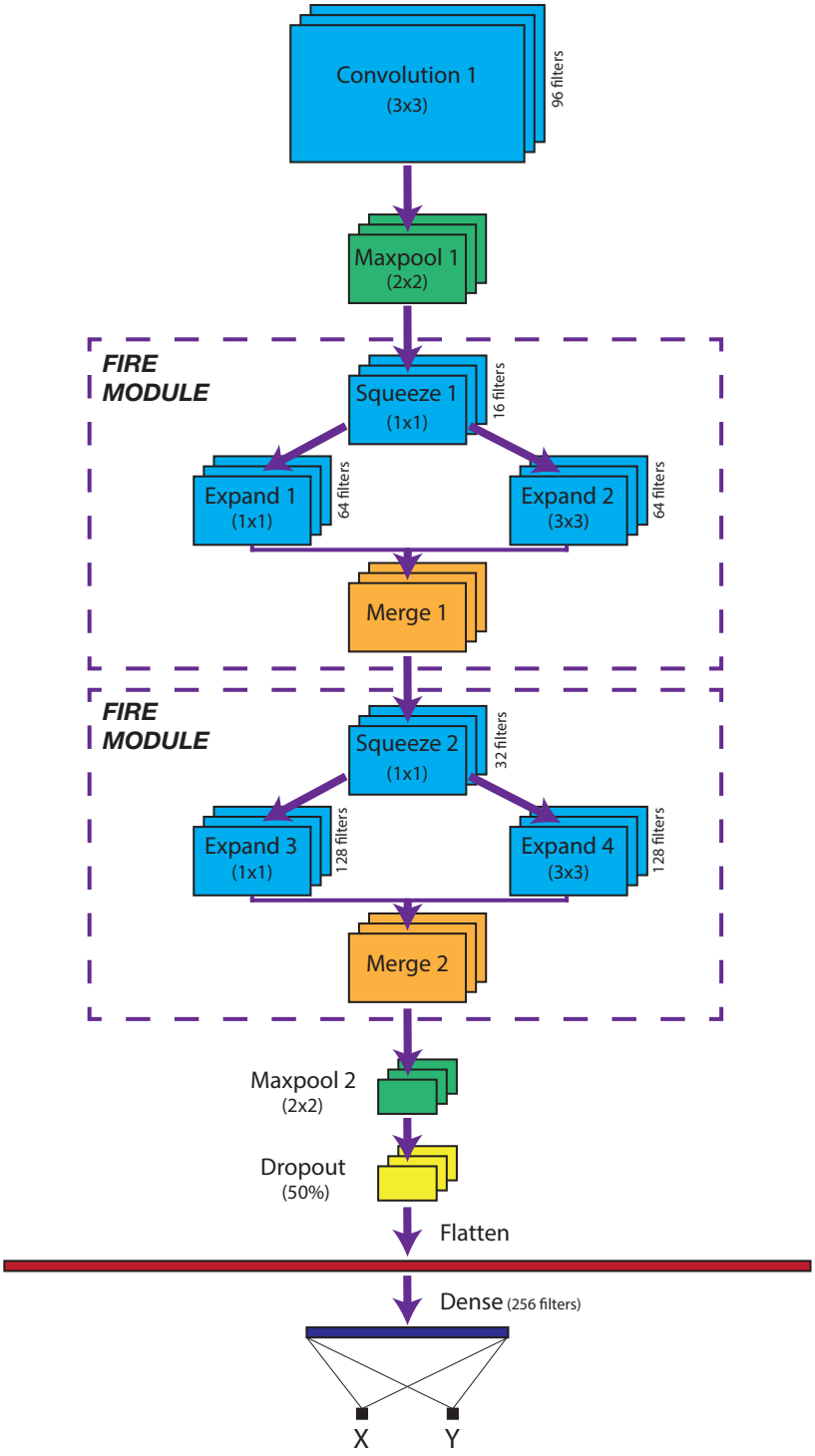


Figure 4.5 – Convolutional neural network architecture used for gaze estimation

4.3 Experimental setup

In our first experiments, we use the same training and evaluation split of the dataset as in the original paper. Under this setup, there are **382315** training examples (left and right eyes combined) and **44996** validation examples. Our first experiment is for comparing the different architectures as explained in the previous chapter, and we run the experiments for each of these 9 models. However, to reduce the runtime of these experiments, we sample the training and validation sets with a **sampling rate of 50%**. This way, we use around ~190k training examples and ~22.5k validation examples for this comparison.

For the second experiment, we use the same training & evaluation split and train both our final model and the baseline system (with multi–modal inputs) on the full dataset for 300 epochs. We report the average gaze estimation error on the complete validation set of ~45k samples for both architectures.

In the last experiment, we run our final network with leave–one–out cross–validation as done in the original work. At each iteration, we set aside the data for one subject from the **427311** dataset samples, and use it as the validation set. In this case, we needed to run the same training & evaluation procedure 15 times (once for each subject), and to reduce the runtime we employed 50% sampling on the training set. We report the validation accuracy in the full validation set to be able to compare our cross–validation results with the baseline (i.e. we use all validation samples for each subject). This way, each training takes around **7.5 hours** on a Tesla K80 GPU, which results in **4.5 days** of GPU usage in total.

4.4 Results

4.4.1 Comparison of Different Architectures

Table 4.1 shows the validation errors for each of the 9 CNN models we listed in the previous sections. We measure the average validation error of the baseline system to be **3.03**° (using 50% of the training and validation sets).

The idea of using very **deep networks** inspired by VGG network have failed in this case, with above **9**° error with no signs of convergence after 40 epochs. We believe that this architecture may have been an overkill for a problem with small input images, and the deep network failed to learn quickly as in other tried architectures.

The network making use of **inception modules** (and some other variants of this network not listed here) could get close to the baseline network's results (after just 60 epochs); however, its validation error was jumping too much between epochs

and we decided to stop the training at this point.

The first architecture making use of **fire modules** showed great performance and decreased the validation error by 17% in this experiment. We believe that the compact nature of these fire modules were suitable for this problem, and the branching & concatenation in each module enabled the network to shuffle and combine the results of previous convolutions in each level. With the success of this module, we analyzed several variants of this network.

In the first two variants, we added **L1 or L2 regularization** to decrease the gap between training and validation errors. In the original v1 network's results, this difference could exceed 1° after 200 epochs, signalling an overfitting (despite the dropout layer). However, these regularizations did not improve the validation performance as seen in the results table. Moreover, the L2 regularization caused a *spiky* validation curve, with huge jumps of up to 0.20° between consecutive epochs.

Another variation of this architecture was adding the **head pose** information at the last stage of the neural network (just before gaze estimation), as in the original work. In this case, this additional input did not improve the performance after 100 epochs, and we decided to stop this experiment. As the eye images are already transformed to look as if they were captured from a frontal face pose, and from a fixed distance with a camera with the same properties, we believe this information didn't provide much value to the network.

Version 2 of this network adds a duplicate of the second fire module, in order to increase model depth and complexity. After 200 epochs, the network achieved exactly the same minimum validation error as in the original model, and we decided to continue with the simpler model for the rest of the experiments.

Version 3 also adds more complexity to the original model with fire modules, increasing the number of convolution filters in each fire module by 25%. We have seen that this did not improve the performance in 200 epochs, and again, we continued with the simpler model.

4.4.2 Comparison Against the Baseline

In the second set of experiments, we train both our system and the baseline system (with multi–modal inputs) on the full training set and check the validation error. The original work did not specify the error rate under this setup, and we measured it to be **3.03**° after 300 epochs. Our network achieved **2.73**° lowest validation error, lowering the error rate by 10% compared to the baseline. Figure 4.6 shows the error distributions for both systems. Here, each histogram bin denotes the number of validation samples that had an error in the corresponding interval.

As seen from the figure, the difference in the *long tail* can be seen clearly, with the baseline having more cases with > 4° errors (**~12k** vs. **~9k** samples).

Table 4.1 – Comparison of minimum validation errors for 9 CNN architectures analyzed in this chapter. If not noted otherwise, the models were run for 200 epochs with a batch size of 128.

Architecture	Val. Error	Comments
Baseline (multi-modal input)	3.03°	
Deep network (VGG-like)	> 9.00°	Not converging at all in 40 epochs
Inception modules	3.31°	Not converging, killed after 50 epochs
Fire modules v1	2.51°	Final architecture
Fire modules v1 (L1 reg.)	3.01°	
Fire modules v1 (L2 reg.)	3.14°	Very "spiky" validation error curve
Fire modules v1 (m.m. input)	2.64°	No improvement, killed after 100 epochs
Fire modules v2	2.51°	Same error, more complex model
Fire modules v3	2.60°	

4.4.3 Leave–One–Out Cross–Validation

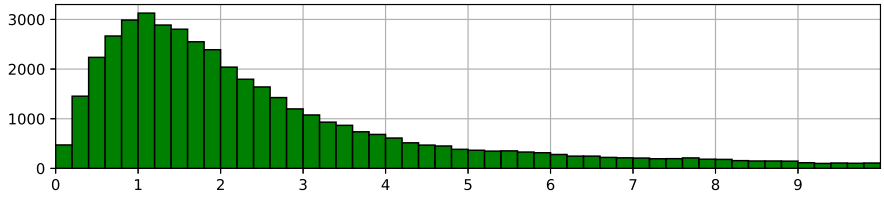
In the leave–one–out cross–validation experiments, we achieve **5.84**° error rate in average for all participants, compared to **6.3**° as reported in the original work. In this experiment too, we have a **7% decrease** in the average error rate. Here, we should note that our networks were trained with **50% of the available training data**, in contrast to the baseline system where all the available data was used during training.

These experiments show that our system can generalize well, and work with better accuracy for never–seen–before subjects.

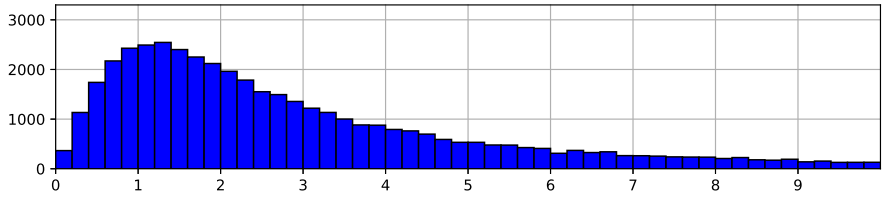
4.4.4 Other Results

To see how our hand–crafted features would fare against the features extracted by the CNN, we extracted our features from the same, large–scale dataset. As in the second experiments explained above, we used the training and evaluation split as in the original work; however, this time we used all the available training data. After training a ridge regressor for mapping to gaze, our feature set yielded **8.37**° error on the whole evaluation set.

After repeating the leave–one–out cross–validation as in the last experiments above, we have seen that our hand–crafted features result in **8.75**° error. This error rate is surprisingly close to the training and evaluation set, and we conclude that our feature representation may not be very suitable for the extracted eye images from this dataset; as the images are not extracted in a stable way. In other words,



(a) Our results



(b) Baseline results

Figure 4.6 – Error distribution for our network compared to the baseline

in our system, we apply anchor point tracking over the video frames to make sure that the point positions remain stable over time; but in the already extracted (called "normalized" by the dataset authors) images, this assumption doesn't hold. With such shifts in eye image extraction steps, our features cannot summarize the eye appearance in a compact way.

4.5 Conclusion

In this last piece of work that we described in this chapter, our aim was to harness the power of the large–scale gaze tracking datasets that are available today. We employed a robust face alignment algorithm, which was combined with a personalized 3D face model to calculate the face pose with better precision, and to extract the subject's eye images in a more reliably.

We topped this face pose estimation algorithm with a convolutional neural network architecture to map the extracted eye images to gaze vectors. Our experiments showed that we improved on the baseline results, lowering the error rates by 7 – 10%. Moreover, we also conclude that using the face pose data in a multi–modal CNN didn't improve the results of our architecture. This may be an indicator that we still haven't discovered the best way to make use of this data, in order to trim a little bit

more of the errors.

Deep learning networks have been revolutionizing all fields of computer vision in the recent years, and we are also experiencing the same change in gaze tracking, although it is still in its early stages. We believe the next few years will see more great examples of work, be it in novel network architectures or larger datasets allowing for more complex training.

5 System

In the previous chapters, we first analyzed all previous works in natural light gaze estimation field, and then proposed several methods with different characteristics. Our first proposal was an appearance–based gaze estimation method, with several improvements over the baseline that provided robustness and ease–of–use for the final system. Later on, we described a method that makes use of compact feature representations for feature–based gaze estimation.

Lastly, we defined our convolutional neural network architecture which can harness the power of large–scale gaze estimation datasets during training, and can generalize well to estimate the gaze of subjects for which it never received training data.

All these works had the aim of building an easy–to–use, cheap eye–tracker option that could open up many potential application areas. In this chapter, we talk about other contributions that we made while working on these projects, which are either focused on providing a novel use–case, or on improving another aspect of the developed system.

In the first section, we define our hardware setup for a prototype of cheap, standalone eye–tracker that makes use of off–the–shelf components. Later on, we propose a way of using similar gaze trackers for an interaction in a multi–display setup. Lastly, we explain our works towards the paralellization of the computations in two components of our eye–tracker software.

With these contributions, we build *our System* which is the combination of our software modules, the hardware prototype, and the use–cases that we propose.

5.1 Cheap Eye–Tracker Prototype

For the software that we worked on in this thesis, we also built a prototype hardware setup that can be used as a standalone device. We believe that this device proposal is a first step towards encapsulating these software systems as a blackbox where the gaze estimation output can be used for other purposes.

Raspberry Pi is a small computer, with a footprint similar to the size of a credit card. It has a relatively powerful ARM processor, for a cheap price of around $40. These specifications make it a suitable choice for our hardware tracker, where

Figure 5.1 – Our hardware setup for the cheap eye–tracker prototype.

the cost–effectiveness is one of the key concerns. Moreover, this tiny computer is capable of running Raspbian operating system, which is based on Debian and provides a familiar runtime environment.

Our full hardware setup consists of a Raspberry Pi device connected to a regular webcam as seen in Figure 5.1.

On top of this hardware, we first install Raspbian and the required libraries such as OpenCV, ffmpeg, etc. This way, we have all the requirements for running our eye–tracker on this device. We use OpenCV to access the connected camera, and output the calculated gaze estimation through a network port. The overview of this software structure is shown in Figure 5.2.

In our prototype, the system is first connected to an external display and it is calibrated by following the gaze target while it moves to different locations on the screen in a grid pattern. Once the calibration is done, the eye–tracker continues calculating the gaze of the subject on this external display, and outputs the estimations on a network port. This way, the users can disconnect the eye–tracker device from the display, and keep using their computer while accessing the gaze estimations by listening to the specific port of the eye–tracker device.

The eye–tracker device that is built this way had a refresh rate of **3Hz** (as we measured 4 years ago). Here, accessing the camera image was one of the bottlenecks, as we measured the speed of only this operation to be around **7Hz** in the early versions of Raspberry Pi machines. We believe that the hardware that is available for

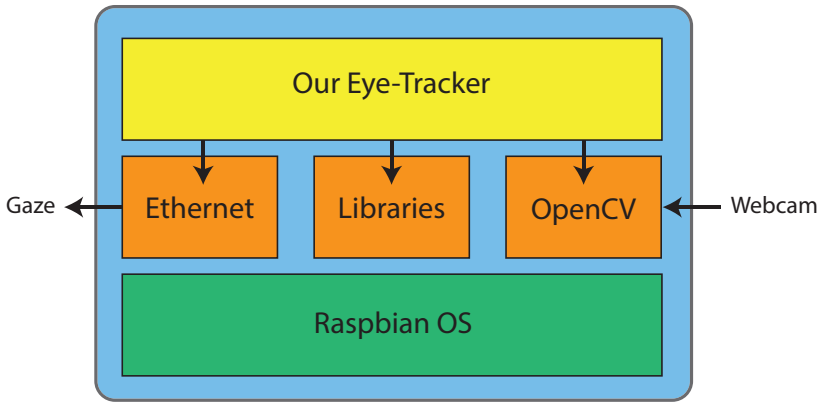


Figure 5.2 – Our software setup for the cheap eye–tracker prototype.

the same price today can provide better speeds, and increase the overall usability of the entire system.

5.2 Multi–Display Interaction Prototype

As one use–case of a cheap, natural light eye–tracker, we designed an interaction that makes use of multiple displays, where other input modalities such as hand gestures and touch gestures are also considered.

As a potential hardware platform, we initially proposed the idea for use with Google Glass devices. These devices have the form of regular eye–glasses, and they have a small display on their right side. The discontinuation of this product proved that it wasn't ready for wide adoption by regular computer users, but we argued that it may be more useful and valuable in niche setups such as interacting with physical documents and digital information in an immersive way.

We propose the fusion of data obtained from our low cost eye–tracker device and the Google Glass camera in order to provide the Goggle Glass display with specific information about the part of the physical document that is being observed. One possible implementation area of this approach lies in the context of an augmented–reading experience: information provided on the display will be changing on–the–fly depending on the word that is being read or the information being analyzed.

For this prototype, first we install our eye–tracking software on the user's desktop computer or on the standalone eye–tracker prototype as described in the previous section. After calibration, we have the information of where the subject is looking

at on the main screen or on the scene (by mapping the gaze to Google Glass camera axis), and we use this to discover information related to the gaze object. The information can be in several forms:

- **Information related to places**, in case the user is analyzing photos of known places (such as touristic locations). This could be information from sources such as Wikipedia, or a gallery of images coming from a web search.
- **Actionable cards**, which provide a quick link to take action about the viewed topic. For example, in case the user is checking out news for a football game, the display could provide a quick link to buy the game's tickets.
- **Map information**, which can help the user to navigate to the location related to the viewed object. For example, in case the user is analyzing a touristic location, the Glass display can show the navigation directions for that place.

The final interface prototype for this system can be seen in Figure 5.3. Here, the left half of the screen shows the gaze targets, which are photos of known locations for this purpose. Each of these targets have some related information assigned to them. For example, the Camp Nou stadium is assigned to a quick link to buy game tickets, whereas the targets for the touristic locations may correspond to gallery of images, navigation directions, etc. Depending on the viewed target, the Google Glass display (shown on the top right of the prototype figure) shows the related information.

For demo purposes, the internal view of the eye–tracker is also shown on this prototype, and is found at the bottom right of the figure.

5.3 GPU–Accelerated Components for Eye-Tracking

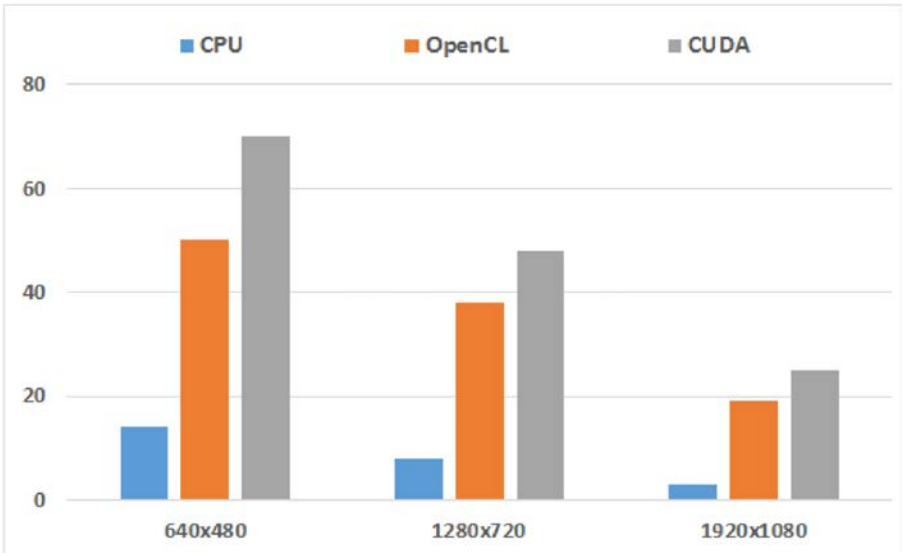
During the eNTERFACE'14 workshops, we worked on GPU–acceleration for some of the eye–tracker modules, together with Sidi Ahmed Mahmoudi.

The proposed implementation can exploit both NVIDIA and ATI graphic cards, based on CUDA4 and OpenCL5. The CUDA version consists of selecting a number of GPU threads so that each thread can perform its processing on one or a group of pixels in parallel. Otherwise, the OpenCL implementation is based on the same process, but using a specific syntax related to OpenCL. The main advantage of OpenCL is its compatibility with both NVIDIA and ATI graphic cards, as it was proposed as a standard for GPU programming. However, CUDA, which allows to program NVIDIA cards only, offers better performances thanks to its adapted programming architecture.

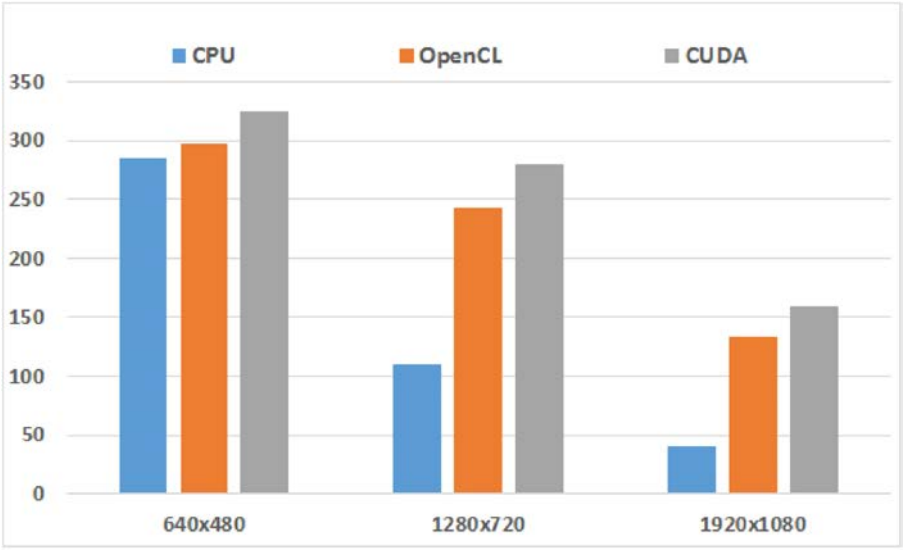
Figure 5.3 – The proposed multi–display interaction prototype. The left side of the screen shows the gaze targets, which trigger related information on the Google Glass display (on top right). The system window can be seen on the bottom right, with debug information such as anchor points and extracted eye images.

Figure 5.4 below compares performance between CPU, CUDA and OpenCL implementations (in terms of fps) of points (eyes, nose and frontal face) detection and optical flow-based tracking. These accelerations allowed to improve the process of webcam-based eye tracker with a factor of 3x. As result, our GPU-based method allows real time eyes tracking with high definition videos.

With this work, we explored the possibilities of making use of the device's GPU to accelerate the processing pipeline of our eye–tracker.



(a) Anchor point selection



(b) Optical–flow point tracking

Figure 5.4 – Acceleration provided by each GPU paralellization method for anchor point selection and point tracking components. The results are shown in FPS rate.

6 Conclusions

6.1 Overview

In this PhD dissertation we have addressed the problem of gaze estimation with visible light cameras in a step–by–step fashion. We built on the same codebase, and improved the gaze tracker with each contribution. Here, we take the opportunity to summarize the findings of this work.

In the first part of this dissertation we started with reviewing the previous works on the field. We analyzed all the works in an exhaustive manner, comparing them from many perspectives such as how they are calibrated, what types of algorithms they employ for gaze estimation, whether they allow for head movements, etc. We also compared their reported performances, to give a better view of which idea has worked better and what are the shortcomings of other methods. Moreover, we reviewed the state–of–the–art in several related topics such as what are the datasets that can be used for measuring the performance of these methods, the services that are providing webcam based gaze tracking for scalable eye tracking studies and so on. Here, our aim was to create an extensive map of the field as a reference for future studies.

In Chapter 3, we analyze the factors that could affect a gaze tracker's performance. We aim at understanding these problematic factors better, in order to take them into account in future work and provide solutions. Moreover, we propose our first gaze tracking system which uses an appearance–based method for estimation. With extensive experiments, we demonstrate how face poses (or camera position), subject distance, variance between subjects, face stability, camera resolution, etc. may hinder the estimation performance. We release the dataset recorded during these experiments as yet another contribution.

In the second part of Chapter 3, we propose a feature–based gaze estimation method. In our review in Chapter 2, we had shown that feature–based gaze estimations are gaining more popularity in the recent years, due to their potential for exploitation. In this work, we propose a new way of encoding the eye region image into meaningful features, which can later on be made head pose invariant by

applying a correction for the pose.

Lastly, in the previous chapter, we proposed our final gaze tracker which starts with module for estimating the face pose, initially using a generic 3D face model which is built with consideration of the variations of facial features among the population. This generic model is customized for the subject during calibration, so that the face pose estimation is more accurate. We connect this component with a deep convolutional neural network, which can leverage the large datasets made available in the recent years. With these two modules, we achieve the head pose free gaze estimation that we aimed at while beginning this thesis.

6.2 Contributions

In this PhD dissertation we have made both practical and theoretical contributions to gaze tracking with regular cameras. We can quickly summarize our contributions as:

- A complete review of the state–of–the–art in the field
- An analysis of factors that may affect gaze tracking with regular webcams
- A public dataset for visible light gaze tracking
- Early stand–alone cheap eye tracker prototype which consists of our software installed on a Raspberry Pi device connected to a webcam
- A novel feature–based gaze estimation method, which proposes new types of features that can describe the eye appearance in a compact representation
- Gaze based interaction prototype for Google Glass devices
- A head pose invariant gaze estimation method that handles variations in the face appearance of different subjects

Apart from these contributions in the main line of research, we also worked on the field of human–computer interaction (HCI), with our works with **Dr. Dan Norton**. In these works, we built multi–display interfaces where one of the displays is also a touch screen. We designed several touch based interactions to enable easy access to large public image datasets such as propaganda posters from the Spanish civil war. We employed our experience from computer vision and used these techniques to add flair to the final installation, making library visits more fun to adults and children alike. Moreover, we also connected this system with sound recordings from the daily life such as school yards, birds, etc., and added yet

another level of connectivity between the sounds and the images. In this case, the computer vision based special effects that we add on the original poster images react to the change in the sound recordings. Apart from scientific conferences, we also presented this work in many cases:

- An installation at the Living Lab in the Library of Miquel Batllori in Sant Cugat, Barcelona 2015-2016
- Demo installation at the Sonar+D event in 2015
- Installation in a Barcelona library for being one of the winners of Mobile Week Barcelona contest in 2017

6.3 Scientific Articles

This dissertation has led to the following communications:

6.3.1 Submitted Journals

- **Onur Ferhat**, & Fernando Vilariño. (2016). "Low Cost Eye Tracking: The Current Panorama " . Computational Intelligence and Neuroscience, , Article ID 8680541.
- **Onur Ferhat**, Fernando Vilariño, & F. Javier Sanchez. (2014). A cheap portable eye-tracker solution for common setups. Journal of Eye Movement Research, 7(3), 1–10.

6.3.2 International Conferences

- Fernando Vilariño, Dan Norton, & **Onur Ferhat**. (2016)." The Eye Doesn't Click – Eyetracking and Digital Content Interaction" In 4S/EASST
- Dan Norton, Fernando Vilariño, & **Onur Ferhat**. (2015)." Memory Field – Creative Engagement in Digital Collections" In Internet Librarian International Conference.
- Fernando Vilariño, Dan Norton, & **Onur Ferhat**. (2015)." Memory Fields: DJs in the Library" In 21 st Symposium of Electronic Arts.
- **Onur Ferhat**, Arcadi Llanza, & Fernando Vilariño. (2015). "Gaze interaction for multi-display systems using natural light eye-tracker " In Proceedings of 2nd International Workshop on Solutions for Automatic Gaze Data Analysis, SAGA 2015.

- **Onur Ferhat**, Arcadi Llanza, & Fernando Vilariño. (2015). "A Feature-Based Gaze Estimation Algorithm for Natural Light Scenarios " In Pattern Recognition and Image Analysis, Proceedings of 7th Iberian Conference , ibPRIA 2015 (Vol. 9117, pp. 569–576). Springer International Publishing.
- **Onur Ferhat**, & Fernando Vilariño. (2013). "A Cheap Portable Eye-Tracker Solution for Common Setups " In 17th European Conference on Eye Movements.

6.3.3 Workshops

- Christian Frisson, Nicolas Riche, Antoine Coutrot, Charles-Alexandre Delestage, Stéphane Dupont, **Onur Ferhat**, Nathalie Guyader, Sidi Ahmed Mahmoudi, Matei Mancas, Parag K. Mital, Alicia Prieto Echániz, François Rocca, Alexis Rochette, & Willy Yvart. (2014). Auracle: how are salient cues situated in audiovisual content? In eNTERFACE Workhshop 2014.

6.4 Contributed Code and Datasets

- **OpenGazer**: The complete eye tracker that we developed during this thesis, which is based on the open source OpenGazer project. We revamped the code to make it completely modular, and in each work, we created a couple of new modules which can be plugged via a simple configuration XML. http://github.com/tiendan/opengazer
- **War Posters**: The web application we developed for multi–display interaction with a touch interface for the Spanish civil war posters project. https://github.com/tiendan/warposters
- **CVC Eye Tracker DB**: A dataset for gaze estimation which includes 48 videos from 12 participants. http://mv.cvc.uab.es/projects/eye-tracker/cvceyetrackerdb

6.4.1 Awards

- **Google Faculty Research Award**: Awarded to Dr. Fernando Vilariño for his proposal about the multi–display interaction that was part of this thesis.

6.4.2 Demos

- *Feature–based gaze estimation* at the Iberian Conference on Pattern Recognition and Image Analysis, Santiago de Compostela, Spain, 2015

Bibliography

[1] *Gaze Direction Estimation Based on Natural Head Movements*, 2007.

[2] EyeTrackShop Website. http://www.eyetrackshop.com, 2017. [Online; accessed 1-June-2017].

[3] NetGazer. http://sourceforge.net/projects/netgazer/, 2017. [Online; accessed 1-June-2017].

[4] The Eye Tribe. http://theeyetribe.com/, 2017. [Online; accessed 1-June-2017].

[5] Tobii AB. Tobii eye-trackers. http://www.tobii.com/, 2017. [Online; accessed 1-June-2017].

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[7] Fares Alnajar, Theo Gevers, Roberto Valenti, and Sennay Ghebreab. Calibration-free gaze estimation using human gaze patterns. In *ICCV*, pages 137–144. IEEE, 2013.

[8] Stylianos Asteriadis, Dimitris Soufleros, Kostas Karpouzis, and Stefanos Kollias. A natural head pose and eye gaze dataset. In *Proceedings of the International Workshop on Affective-Aware Virtual Agents and Social Robots*, AFFINE '09, pages 1:1–1:4, New York, NY, USA, 2009. ACM.

[9] Stylianos Asteriadis, Dimitris Soufleros, Kostas Karpouzis, and Stefanos Kollias. Head Pose and Eye Gaze (HPEG) Dataset.

http://sspnet.eu/2010/02/head-pose-and-eye-gaze-hpeg-dataset/, 2017. [Online; accessed 1-June-2017].

[10] Bäck, David. *Neural Network Gaze Tracking using Web Camera*. PhD thesis, Linköping University, December 2005.

[11] Seung Jin Baek, Kang a. Choi, Chunfei Ma, Young Hyun Kim, and Sung Jea Ko. Eyeball model-based iris center localization for visible image-based eye-gaze tracking systems. *IEEE Transactions on Consumer Electronics*, 59(0098):415–421, 2013.

[12] Shumeet Baluja and Dean Pomerleau. Non-intrusive gaze tracking using artificial neural networks. *Advances in Neural Information Processing Systems*, (January):1–14, 1994.

[13] Michael Calonder, Vincent Lepetit, Mustafa Özuysal, Tomasz Trzcinski, Christoph Strecha, and Pascal Fua. Brief: Computing a local binary descriptor very fast. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(7):1281–1298, 2012.

[14] Modesto Castrillón-Santana. Modesto Castrillón-Santana, 2013.

[15] Jixu Chen and Qiang Ji. 3D gaze estimation with a single camera without IR illumination. In *ICPR*, pages 1–4. IEEE, 2008.

[16] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[17] Stefania Cristina and Kenneth P. Camilleri. Model-based head pose-free gaze estimation for assistive communication. *Computer Vision and Image Understanding*, 149:157–170, 2016.

[18] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR (1)*, pages 886–893. IEEE Computer Society, 2005.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[20] Andrew T. Duchowski. *Eye tracking methodology – theory and practice*. Springer, 2003.

[21] Onur Ferhat, Arcadi Llanza, and Fernando Vilariño. A feature-based gaze estimation algorithm for natural light scenarios. In *IbPRIA*, pages 569–576, 2015.

[22] Onur Ferhat and Fernando Vilariño. CVC Eye Tracker. https://github.com/tiendan/OpenGazer, 2017. [Online; accessed 1-June-2017].

[23] Onur Ferhat and Fernando. Vilariño. CVC Eye Tracking DB. http://mv.cvc.uab.es/projects/eye-tracker/cvceyetrackerdb, 2017. [Online; accessed 1-June-2017].

[24] Onur Ferhat, Fernando Vilariño, and Francisco Javier Sánchez. A Cheap Portable Eye-tracker Solution for Common Setups. *Journal of Eye Movement Research*, 7(3):1–10, 2014.

[25] Andrew W. Fitzgibbon, Maurizio Pilu, and Robert B. Fisher. Direct least squares fitting of ellipses. In *ICPR*, pages 253–257. IEEE Computer Society, 1996.

[26] Takashi Fukuda, Kosuke Morimoto, and Hayato Yamana. Model-Based Eye-Tracking Method for Low-Resolution Eye-Images. In *2nd Workshop on Eye Gaze in Intelligent Human Machine Interaction*, 2011.

[27] Kenneth Alberto Funes Mora, Florent Monay, and Jean-Marc Odobez. EYEDIAP: A Database for the Development and Evaluation of Gaze Estimation Algorithms from RGB and RGB-D Cameras. In *Proceedings of the ACM Symposium on Eye Tracking Research and Applications*. ACM, March 2014.

[28] Kenneth Alberto Funes Mora, Florent Monay, and Jean-Marc Odobez. EYEDIAP Dataset. http://www.idiap.ch/dataset/eyediap, 2017. [Online; accessed 1-June-2017].

[29] SensoMotoric Instruments GmbH. Eye tracking solutions by smi. http://www.smivision.com/, 2017. [Online; accessed 1-June-2017].

[30] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J., 2008.

[31] Shameem Hameed. Haar cascade for eyes. http://www-personal.umich.edu/2017. [Online; accessed 1-June-2017].

[32] Dan Witzner Hansen, John Paulin Hansen, Mads Nielsen, Anders Sewerin Johansen, and Mikkel B. Stegmann. Eye typing using Markov and Active Appearance Models. In *WACV*, pages 132–136. IEEE Computer Society, 2002.

[33] Dan Witzner Hansen and Qiang Ji. In the eye of the beholder: A survey of models for eyes and gaze. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(3):478–500, 2010.

[34] Dan Witzner Hansen, Mads Nielsen, John Paulin Hansen, Anders Sewerin Johansen, and Mikkel B. Stegmann. Tracking eyes using shape and appearance. In *MVA*, pages 201–204, 2002.

[35] Dan Witzner Hansen and Arthur E. C. Pece. Eye typing off the shelf. In *CVPR (2)*, pages 159–164, 2004.

[36] Dan Witzner Hansen and Arthur E. C. Pece. Eye tracking in the wild. *Computer Vision and Image Understanding*, 98(1):155–181, 2005.

[37] Qiuhai He, Xiaopeng Hong, Xiujuan Chai, Jukka Holappa, Guoying Zhao, Xilin Chen, and Matti Pietikäinen. OMEG: Oulu multi-pose eye gaze dataset. In Rasmus R. Paulsen and Kim Steenstrup Pedersen, editors, *SCIA*, volume 9127 of *Lecture Notes in Computer Science*, pages 418–427. Springer, 2015.

[38] Marko Heikkilä, Matti Pietikäinen, and Cordelia Schmid. Description of interest regions with center-symmetric local binary patterns. In Prem Kumar Kalra and Shmuel Peleg, editors, *ICVGIP*, volume 4338 of *Lecture Notes in Computer Science*, pages 58–69. Springer, 2006.

[39] Tom Heyman, Vincent Spruyt, and Alessandro Ledda. 3D Face Tracking and Gaze Estimation Using a Monocular Camera. In *2nd International Conference on Positioning and Context-Awareness (PoCA 2011)*, pages 23–28, 2011.

[40] Corey Holland and Oleg V. Komogortsev. Eye tracking on unmodified common tablets: challenges and solutions. In Carlos Hitoshi Morimoto, Howell O. Istance, Stephen N. Spencer, Jeffrey B. Mulligan, and Pernilla Qvarfordt, editors, *ETRA*, pages 277–280. ACM, 2012.

[41] Corey Holland and Oleg V. Komogortsev. Neural Network Eye Tracker (NNET). http://cs.txstate.edu/ ok11/nnet.html, 2017. [Online; accessed 1-June-2017].

[42] Qiong Huang, Ashutosh Sabharwal, and Ashok Veeraraghavan. Rice TabletGaze Dataset. http://www.sh.rice.edu/tablet_gaze.html, 2017. [Online; accessed 1-June-2017].

[43] Qiong Huang, Ashok Veeraraghavan, and Ashutosh Sabharwal. TabletGaze: A dataset and baseline algorithms for unconstrained appearance-based gaze estimation in mobile tablets. *CoRR*, abs/1508.01244, 2015.

[44] Shang-Che Huang, Yi-Leh Wu, Wei-Chih Hung, and Cheng-Yuan Tang. Point-of-regard measurement via iris contour with one eye from single image. In *ISM*, pages 336–341. IEEE Computer Society, 2010.

[45] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[46] GazeHawk Inc. Gazehawk. http://www.gazehawk.com, 2017. [Online; accessed 1-June-2017].

[47] Qualcomm Technologies Inc. Snapdragon SDK for Android. https://developer.qualcomm.com/software/snapdragon-sdk-android, 2017. [Online; accessed 1-June-2017].

[48] Ibrahim Ince and Jin Woo Kim. A 2d eye gaze estimation system with low-resolution webcam images. *EURASIP J. Adv. Sig. Proc.*, 2011:40, 2011.

[49] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[50] Takahiro Ishikawa, Simon Baker, Iain Matthews, and Takeo Kanade. Passive driver gaze tracking with active appearance models. In *Proceedings of the 11th World Congress on Intelligent Transportation Systems*, volume 3, 2004.

[51] László A. Jeni and Jeffrey F. Cohn. Person-independent 3d gaze estimation using face frontalization. In *CVPR Workshops*, pages 792–800. IEEE Computer Society, 2016.

[52] Tilke Judd, Krista A. Ehinger, Frédo Durand, and Antonio Torralba. Learning to predict where humans look. In *ICCV*, pages 2106–2113. IEEE, 2009.

[53] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *CVPR*, pages 1867–1874. IEEE Computer Society, 2014.

[54] Eun Yi Kim, Sin Kuk Kang, Keechul Jung, and Hang Joon Kim. Eye mouse: mouse implementation using eye tracking. In *Consumer Electronics, 2005. ICCE. 2005 Digest of Technical Papers. International Conference on*, pages 207 – 208, jan. 2005.

[55] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[56] Davis E. King. dlib c++ library, 2017. [Online; accessed 1-June-2017].

[57] Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra M. Bhandarkar, Wojciech Matusik, and Antonio Torralba. Eye tracking for everyone. In *CVPR*, pages 2176–2184. IEEE Computer Society, 2016.

[58] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[60] Chih-Chuan Lai, Yu-Ting Chen, Kuan-Wen Chen, Shen-Chi Chen, Sheng-Wen Shih, and Yi-Ping Hung. Appearance-Based Gaze Tracking with Free Head Movement. In *ICPR*, volume 1, pages 1869–1873, 2014.

[61] Ke Liang, Charles Tijus, Youssef Chahir, François Jouen, and Michèle Molina. Appearance-Based Gaze Tracking with Spectral Clustering and Semi-Supervised Gaussian Process Regression. In *Eye Tracking South Africa*, volume 1, pages 29–31, 2013.

[62] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[63] Feng Lu and Xiaowu Chen. Person-independent eye gaze prediction from eye images using patch-based features. *Neurocomputing*, 182:10–17, 2016.

[64] Feng Lu, Takahiro Okabe, Yusuke Sugano, and Yoichi Sato. A Head Pose-free Approach for Appearance-based Gaze Estimation. In *BMVC*, pages 126.1–126.11, 2011.

[65] Feng Lu, Takahiro Okabe, Yusuke Sugano, and Yoichi Sato. Learning gaze biases with head motion for head pose-free gaze estimation. *Image Vision Comput.*, 32(3):169–179, 2014.

[66] Feng Lu, Yusuke Sugano, Takahiro Okabe, and Yoichi Sato. Inferring human gaze from appearance via adaptive linear regression. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, editors, *ICCV*, pages 153–160. IEEE, 2011.

[67] Feng Lu, Yusuke Sugano, Takahiro Okabe, and Yoichi Sato. Head pose-free appearance-based gaze sensing via eye image synthesis. In *ICPR*, number Icpr, pages 1008–1011, 2012.

[68] Feng Lu, Yusuke Sugano, Takahiro Okabe, and Yoichi Sato. Adaptive Linear Regression for Appearance-Based Gaze Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(10):2033–2046, 2014.

[69] Feng Lu, Yusuke Sugano, Takahiro Okabe, and Yoichi Sato. Gaze Estimation From Eye Appearance: A Head Pose-Free Method via Eye Image Synthesis. *IEEE Transactions on Image Processing*, 24(11):3680–3693, nov 2015.

[70] Francis Martinez, Andrea Carbone, and Edwige Pissaloux. Gaze estimation using local features and non-linear regression. In *IEEE International Conference on Image Processing (ICIP'12)*, volume 1, pages 1961–1964, 2012.

[71] Christopher D. McMurrough, Vangelis Metsis, Dimitrios Kosmopoulos, Ilias Maglogiannis, and Fillia Makedon. A dataset for point of gaze detection using head poses and eye images. *Journal on Multimodal User Interfaces*, 7(3):207–215, 2013.

[72] Yiu ming Cheung and Qinmu Peng. Eye gaze tracking with a web camera in a desktop environment. *IEEE T. Human-Machine Systems*, 45(4):419–430, 2015.

[73] Mohammad Reza Mohammadi and Abolghasem Raie. Robust Pose-Invariant Eye Gaze Estimation Using Geometrical Features of Iris and Pupil Images. In *20th Iranian Conference on Electrical Engineering (ICEE)*, pages 593–598, 2012.

[74] Carlos Hitoshi Morimoto and Marcio R. M. Mimica. Eye gaze tracking techniques for interactive applications. *Computer Vision and Image Understanding*, 98(1):4–24, 2005.

[75] Susan M. Munn and Jeff B. Pelz. 3D point-of-regard, position and head orientation from a portable monocular video-based eye tracker. In *ETRA*, pages 181–188. ACM, 2008.

[76] Neurotechnology. SentiGaze SDK. http://www.neurotechnology.com/sentigaze.html, 2017. [Online; accessed 1-June-2017].

[77] Ba Linh Nguyen. Eye gaze tracking. In *RIVF*, pages 1–4. IEEE, 2009.

[78] Ba Linh Nguyen, Youssef Chahir, Michèle Molina, Charles Tijus, and François Jouen. Eye gaze tracking with free head movements using a single camera. In Nguyen Trong Giang, Nguyen Thuc Hai, and Huynh Quyet Thang, editors, *SoICT*, volume 449 of *ACM International Conference Proceeding Series*, pages 108–113. ACM, 2010.

[79] PhiBang Nguyen, Julien Fleureau, Christel Chamaret, and Philippe Guillotel. Calibration-free gaze tracking using particle filter. In *ICME*, pages 1–6. IEEE Computer Society, 2013.

[80] S. Nissen. Implementation of a Fast Artificial Neural Network Library (FANN). *Report, Department of Computer Science University of Copenhagen (DIKU)*, 31, 2003.

[81] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, 29(1):51–59, 1996.

[82] Yasuhiro Ono, Takahiro Okabe, and Yoichi Sato. Gaze estimation from low resolution images. *Lecture Notes in Computer Science*, 4319:178–188, 2006.

[83] Victoria Ponz, Arantxa Villanueva, and Rafael Cabeza. Dataset for the evaluation of eye detector for gaze estimation. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 681–684, New York, NY, USA, 2012. ACM.

[84] Victoria Ponz, Arantxa Villanueva, and Rafael Cabeza. Gi4E Database. http://gi4e.unavarra.es/databases/gi4e/, 2017. [Online; accessed 1-June-2017].

[85] Alan Poston. Human engineering design data digest. Technical report, Department of Defense Human Factors Engineering Technical Advisory Group, 2000.

[86] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[87] Michael Reale, Terry Hung, and Lijun Yin. Viewing direction estimation based on 3D eyeball construction for HRI. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 24–31, 2010.

[88] Javier San Agustin and Martin Tall. Gaze tracking library. http://sourceforge.net/projects/gazetrackinglib/, 2017. [Online; accessed 1-June-2017].

[89] Modesto Castrillón Santana, Oscar Déniz-Suárez, Daniel Hernández-Sosa, and Javier Lorenzo. A comparison of face and facial feature detectors based on the viola-jones general object detection framework. *Mach. Vis. Appl.*, 22(3):481–494, 2011.

[90] Timo Schneider, Boris Schauerte, and Rainer Stiefelhagen. Manifold Alignment for Person Independent Appearance-based Gaze Estimation. In *ICPR*, pages 1167–1172, 2014.

[91] Laura Sesma, Arantxa Villanueva, and Rafael Cabeza. Evaluation of pupil center-eye corner vector for gaze estimation using a web cam. In Carlos Hitoshi Morimoto, Howell O. Istance, Stephen N. Spencer, Jeffrey B. Mulligan, and Pernilla Qvarfordt, editors, *ETRA*, pages 217–220. ACM, 2012.

[92] Weston Sewell and Oleg Komogortsev. Real-time eye gaze tracking with an unmodified commodity webcam employing a neural network. In Elizabeth D. Mynatt, Don Schoner, Geraldine Fitzpatrick, Scott E. Hudson, W. Keith Edwards, and Tom Rodden, editors, *CHI Extended Abstracts*, pages 3739–3744. ACM, 2010.

[93] Sightcorp. InSight SDK. http://sightcorp.com/insight/, 2017. [Online; accessed 1-June-2017].

[94] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[95] Evangelos Skodras, Vasileios G. Kanas, and Nikolaos D. Fakotakis. On visual gaze tracking based on a single low cost camera. *Sig. Proc.: Image Comm.*, 36:29–42, 2015.

[96] Brian A. Smith, Qi Yin, Steven K. Feiner, and Shree K. Nayar. Gaze locking: Passive eye contact detection for human-object interaction. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 271–280, New York, NY, USA, 2013. ACM.

[97] Brian A. Smith, Qi Yin, Steven K. Feiner, and Shree K. Nayar. Columbia Gaze Data Set. http://www.cs.columbia.edu/CAVE/databases, 2017. [Online; accessed 1-June-2017].

[98] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[99] Sticky. Sticky. https://www.sticky.ai/how-it-works, 2017. [Online; accessed 1-June-2017].

[100] Rainer Stiefelhagen, Jie Yang, and Alex Waibel. Tracking Eyes and Monitoring Eye Gaze, 1997.

[101] Yusuke Sugano, Yasuyuki Matsushita, and Yoichi Sato. Appearance-based gaze estimation using visual saliency. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(2):329–341, 2013.

[102] Yusuke Sugano, Yasuyuki Matsushita, and Yoichi Sato. Learning-by-synthesis for appearance-based 3d gaze estimation. In *CVPR*, pages 1821–1828. IEEE, 2014.

[103] Yusuke Sugano, Yasuyuki Matsushita, and Yoichi Sato. Multi-view Gaze Dataset. http://www.hci.iis.u-tokyo.ac.jp/datasets/, 2017. [Online; accessed 1-June-2017].

[104] Yusuke Sugano, Yasuyuki Matsushita, Yoichi Sato, and Hideki Koike. An incremental learning method for unconstrained gaze estimation. In *ECCV (3)*, pages 656–667, 2008.

[105] Yusuke Sugano, Yasuyuki Matsushita, Yoichi Sato, and Hideki Koike. Appearance-Based Gaze Estimation With Online Calibration From Mouse Operations. *IEEE T. Human-Machine Systems*, PP(99):1–11, 2015.

[106] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[107] Kar-Han Tan, David J. Kriegman, and Narendra Ahuja. Appearance-based eye gaze estimation. In *WACV*, pages 191–195. IEEE Computer Society, 2002.

[108] Visage Technologies. FaceTrack - Visage Technologies. http://visagetechnologies.com/products-and-services/visagesdk/facetrack/, 2017. [Online; accessed 1-June-2017].

[109] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[110] Fabian Timm and Erhardt Barth. Accurate eye centre localisation by means of gradients. In *VISAPP 2011 - Proceedings of the Sixth International Conference on Computer Vision Theory and Applications, Vilamoura, Algarve, Portugal, 5-7 March, 2011*, pages 125–130, 2011.

[111] Diego Torricelli, Silvia Conforto, Maurizio Schmid, and Tommaso D'Alessio. A neural-based remote eye gaze tracker under natural head motion. *Computer Methods and Programs in Biomedicine*, 92(1):66–78, 2008.

[112] Diego Torricelli, Michela Goffredo, Silvia Conforto, Maurizio Schmid, and Tommaso D'Alessio. A novel neural eye gaze tracker. In *2nd International Workshop on Biosignal Processing and Classification-Biosignals and Sensing for Human Computer Interface (BPC 2006)*, pages 86–95, 2006.

[113] Kentaro Toyama. "Look, ma–no hands!" Hands-free cursor control with real-time 3D face tracking. In *Workshop Perceptual User Interfaces*, pages 49–54. IEEE Computer Society, 1998.

[114] Umoove. Umoove. http://www.umoove.me/, 2017. [Online; accessed 1-June-2017].

[115] Roberto Valenti, Nicu Sebe, and Theo Gevers. Simple and efficient visual gaze estimation. In *ICMI,MIAUCE Workshop*, number 3, 2008.

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

[117] Roberto Valenti, Jacopo Staiano, Nicu Sebe, and Theo Gevers. Webcam-based visual gaze estimation. In Pasquale Foggia, Carlo Sansone, and Mario Vento, editors, *ICIAP*, volume 5716 of *Lecture Notes in Computer Science*, pages 662–671. Springer, 2009.

[118] Arantxa Villanueva, Victoria Ponz, Laura Sesma-Sanchez, Mikel Ariz, Sonia Porta, and Rafael Cabeza. Hybrid method based on topography for robust detection of iris center and eye corners. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(4):25:1–25:20, August 2013.

[119] P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[120] Alex Wallar. Camgaze. https://github.com/wallarelvo/camgaze, 2017. [Online; accessed 1-June-2017].

[121] Jian-Gang Wang, Eric Sung, and Ronda Venkateswarlu. Eye gaze estimation from a single image of one eye. In *ICCV*, pages 136–143. IEEE Computer Society, 2003.

[122] Jian-Gang Wang, Eric Sung, and Ronda Venkateswarlu. Estimating the eye gaze from one eye. *Computer Vision and Image Understanding*, 98(1):83–103, 2005.

[123] Qiqi Wang. Mir. http://sourceforge.net/projects/mvinterp, 2017. [Online; accessed 1-June-2017].

[124] Qiqi Wang, Parviz Moin, and Gianluca Iaccarino. A high order multivariate approximation scheme for scattered data sets. *Journal of Computational Physics*, 229(18):6343 – 6361, 2010.

[125] U. Weidenbacher, G. Layher, P.-M. Strauss, and H. Neumann. A comprehensive head pose and gaze database. In *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on*, pages 455–458, Sept 2007.

[126] U. Weidenbacher, G. Layher, P.-M. Strauss, and H. Neumann. UUlm Head Pose and Gaze Database. https://www.uni-ulm.de/in/neuroinformatik/mitarbeiter/g-layher/image-databases.html, 2017. [Online; accessed 1-June-2017].

[127] Oliver Williams, Andrew Blake, and Roberto Cipolla. Sparse and Semi-supervised Visual Mapping with the S^3GP. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 230–237, Washington, DC, USA, 2006. IEEE Computer Society.

[128] Adam Wojciechowski and Krzysztof Fornalczyk. Exponentially smoothed interactive gaze tracking method. In Leszek J. Chmielewski, Ryszard Kozera, Bok-Suk Shin, and Konrad W. Wojciechowski, editors, *ICCVG*, volume 8671 of *Lecture Notes in Computer Science*, pages 645–652. Springer, 2014.

[129] Erroll Wood, Tadas Baltrusaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. Learning an appearance-based gaze estimator from one million synthesised images. In Pernilla Qvarfordt and Dan Witzner Hansen, editors, *ETRA*, pages 131–138. ACM, 2016.

[130] Erroll Wood, Tadas Baltrusaitis, Xucong Zhang, Yusuke Sugano, Peter Robinson, and Andreas Bulling. Rendering of eyes for eye-shape registration and gaze estimation. In *ICCV*, pages 3756–3764. IEEE Computer Society, 2015.

[131] Erroll Wood and Andreas Bulling. Eyetab: model-based gaze estimation on unmodified tablet computers. In Pernilla Qvarfordt and Dan Witzner Hansen, editors, *ETRA*, pages 207–210. ACM, 2014.

[132] Erroll Wood and Andreas Bulling. EyeTab. https://github.com/errollw/EyeTab/, 2017. [Online; accessed 1-June-2017].

[133] Haiyuan Wu, Qian Chen, and Toshikazu Wada. Conic-based algorithm for visual line estimation from one image. In *FGR*, pages 260–265. IEEE Computer Society, 2004.

[134] Haiyuan Wu, Yosuke Kitagawa, Toshikazu Wada, Takekazu Kato, and Qian Chen. Tracking iris contour with a 3d eye-model for gaze estimation. In Yasushi Yagi, Sing Bing Kang, In-So Kweon, and Hongbin Zha, editors, *ACCV (1)*, volume 4843 of *Lecture Notes in Computer Science*, pages 688–697. Springer, 2007.

[135] Li Xia, Bin Sheng, Wen Wu, Lizhuang Ma, and Ping Li. Accurate gaze tracking from single camera using gabor corner detector. *Multimedia Tools Appl.*, 75(1):221–239, 2016.

[136] xLabs Pty Ltd. xLabs eye, gaze and head tracking. http://xlabsgaze.com/, 2017. [Online; accessed 1-June-2017].

[137] Li-Qun Xu, Dave Machin, and Phil Sheppard. A novel approach to real-time non-intrusive gaze finding. In John N. Carter and Mark S. Nixon, editors, *BMVC*. British Machine Vision Association, 1998.

[138] Pingmei Xu, Krista A. Ehinger, Yinda Zhang, Adam Finkelstein, Sanjeev R. Kulkarni, and Jianxiong Xiao. Turkergaze: Crowdsourcing saliency with webcam based eye tracking. *CoRR*, abs/1504.06755, 2015.

[139] Pingmei Xu and Jianxiong Xiao. TurkerGaze. https://github.com/PrincetonVision/TurkerGaze, 2017. [Online; accessed 1-June-2017].

[140] Hirotake Yamazoe, Akira Utsumi, Tomoko Yonezawa, and Shinji Abe. Remote and head-motion-free gaze tracking for real environments with automated head-eye model calibrations. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–6, 2008.

[141] Hirotake Yamazoe, Akira Utsumi, Tomoko Yonezawa, and Shinji Abe. Remote gaze estimation with a single camera based on facial-feature tracking without special calibration actions. In *ETRA*, pages 245–250. ACM, 2008.

[142] Pei Yu, Jiahuan Zhou, and Ying Wu. Learning reconstruction-based remote gaze estimation. In *CVPR*, pages 3447–3455. IEEE Computer Society, 2016.

[143] Wen Zhang, Tai-Ning Zhang, and Sheng-Jiang Chang. Gazing estimation and correction from elliptical features of one iris. In *2010 3rd International Congress on Image and Signal Processing*, pages 1647–1652, 2010.

[144] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Appearance-based gaze estimation in the wild. In *CVPR*, pages 4511–4520. IEEE, 2015.

[145] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Mpiigaze dataset. https://www.mpi-inf.mpg.de/departments/computer-vision-and-multimodal-computing/research/gaze-based-human-computer-interaction/appearance-based-gaze-estimation-in-the-wild-mpiigaze/, 2017. [Online; accessed 1-June-2017].

[146] Yanxia Zhang, Andreas Bulling, and Hans Gellersen. Towards pervasive eye tracking using low-level image features. In Carlos Hitoshi Morimoto, Howell O. Istance, Stephen N. Spencer, Jeffrey B. Mulligan, and Pernilla Qvarfordt, editors, *ETRA*, pages 261–264. ACM, 2012.

[147] Jie Zhu and Jie Yang. Subpixel eye gaze tracking. In *Fifth IEEE International Conference on Automatic Face Gesture Recognition*, pages 131–136. IEEE Computer Society, 2002.

[148] Piotr Zieliński. Opengazer: open-source gaze tracker for ordinary webcams (software). http://www.inference.phy.cam.ac.uk/opengazer/, 2017. [Online; accessed 1-June-2017].