EXAMENSARBETE INOM ELECTRONICS AND COMPUTER ENGINEERING,
GRUNDNIVÅ, 15 HP
STOCKHOLM, SVERIGE 2019

Implementing gaze tracking with a simple web camera

MARTIN FALKE

LUCAS HÖGLUND

KTH
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Acknowledgments

To begin with, we would like to thank Johan Montelius and Thomas Sjöland, our examiner and instructor respectively, for extensive guidance with the writing of the thesis report as well as the project work itself. We would also like to extend our gratitude to the Flatbush Zombies for making and releasing music that inspired and motivated us while working. Finally, we would like to thank Gustaf Öqvist Seimyr and Lexplore AB for guiding us throughout the project and for providing the project idea.

Abstract

Gaze tracking is a field within computer vision that has a large number of possible areas of application. However, much of the software available for gaze tracking is locked to licenses and/or certain equipment that can be expensive, hard to get a hold of, intrusive for the user, or simply impractical. The research within gaze tracking typically focuses on the theoretical models used to perform the tracking, rather than the software implementation of it. The purpose of this report is to make real-time gaze tracking systems more easily available to the common user and developer. Hence, this thesis explores a possible implementation of gaze tracking using C# in Visual Studio, with the help of a number of different libraries, only using a simple web camera as hardware. Further, the system is analyzed and evaluated by the precision of the subject's eye movements which is projected on the screen. The resulting system is heavily inaccurate and imprecise but changing and adding a few key software components such as estimating the user's head pose could vastly improve the accuracy and precision of the system.

Keywords: computer vision, gaze tracking, web camera, desktop environment

Sammanfattning

Blickspårning är ett fält inom datorseende som har ett stort antal möjliga tilläpmningsområden. Däremot så är mycket av den tillgängliga mjukvaran låst till licenser och/eller viss utrustning som kan vara dyr, svår att få tag på, påträngande, eller helt enkelt opraktisk. Forskningen inom blickspårning fokuserar typiskt sett på den teoretiska modellen som beskriver hur spårningen går till, snarare än hur det implementeras i mjukvara. Syftet med den här rapporten är att skapa ett blickspårningsprogram som körs i realtid för att underlätta och göra det mer tillgängligt för andra användare och utvecklare. Därför utforskar den här uppsatsen ett sätt att implementera blickspårning med C# i Visual Studio, med hjälp av ett antal olika kodbibliotek, och endast en enkel webbkamera som hårdvara. Systemet kommer även att bli analyserat och evaluerat med hänseende till träffsäkerheten i testpersoners ögonrörelse som projiceras på skärmen. Det resulterande systemet har en oerhört låg träffsäkerhet och exakthet, men genom att ändra några nyckelkomponenter i mjukvaran, så som att uppskatta hur användarens huvud är riktat, så kan systemets träffsäkerhet och precision ökas dramatiskt.

Nyckelord: datorseende, blickspårning, webbkamera, skrivbordsmiljö

Contents

Chapter 1

Introduction

1.1 Background

Gaze tracking is a field that is contained by the broad category of computer vision. Computer vision entails anything that allows a computer to capture images of reality and process it in some way [1]. This can be done in a myriad of ways, by using sensors that collect data about the surroundings through either light or sound. Gaze tracking is a field that concerns itself with, through the usage of some sort of computer vision that is based in sensing light, finding where a user's vision is directed. In other words, gaze tracking is about finding out where eyes are looking.

Gaze tracking can be used for numerous areas of application with a broad variety of purposes. It can be used as a tool for controlling computers either in order to provide a more immersive experience for the user or to allow disabled people to use the computer who otherwise would not be able to. Another area where gaze tracking is useful is in education, where the tracking data can be used to record, evaluate and compare the reading patterns of students. By doing this, one could identify students that have trouble reading and support them with additional resources so that they do not fall behind in their education [2].

1.2 Problem

While there have been a plethora of different implementations of gaze tracking based on various equipment and methods, the accessibility of such implementations is scarce for the common user. Many of the systems that are available either require intrusive equipment, expensive equipment and/or licenses that may not be practical for all areas of application of gaze tracking. There has been a lot of purely theoretical papers on how to perform gaze tracking, that do not touch on the subject of software implementation. How can one implement a gaze tracking software system, using only a simple web camera as well as free and publicly available software resources?

1.3 Purpose

The purpose of this report is to make real-time gaze tracking systems more easily available to the common user and developer. By showing the different design choices and software tools that are required and available to the public, one can use this report as a tool to either fully recreate the described system or as inspiration to create something similar.

1.4 Goal

The goal of this thesis project is to develop a working real-time gaze tracking system. The system must be able to run where the only hardware being used is a web camera that is capable of recording video with a resolution between Full HD and 4k.

1.5 Benefits, ethics and sustainability

There are many possible benefits that may result from improving the accessibility of gaze tracking. Since its possible applications include analyzing a person's reading, a system like the one described in this report could, for instance, provide benefits to education by adapting teaching approaches to individual students' test results. While there are many areas of application that can be of benefit to society, there are also ways to abuse the technology. In a dystopian society, gaze tracking is used to control and surveil the people. This would be a grave infringement on the personal integrity of the people and could contribute to the power of a totalitarian government. Despite this potential abuse of the technology, contributing to the common person's understanding of gaze tracking can decrease the room for abuse, as people realize the power of the technology and consequently actively resist its abuse.

An aspect of sustainability exists when considering the development and usage of gaze tracking systems. All forms of digitization inevitably lead to further dependency on electricity. Thus, one ought to be careful when deciding what applications gaze tracking systems are used for. One might want to avoid applications where a sudden shutdown of the system can have disastrous consequences.

1.6 Methodology

In this report, the method consists of a literature study, a method for estimating the user's gaze, the choice of development environment, and an evaluation method to analyze and evaluate the system itself.

The literature study is for finding relevant papers, reports, and information regarding gaze tracking so that differentiating between high-quality gaze tracking methods and choosing one becomes doable. The chosen method ought to fulfill the purpose and goal of this thesis. That is, a simple implementation of a real-time gaze tracking system which only makes use of a single web camera and also provides an understandable description of how the implementation was done.

The method for estimating the user's gaze is the theoretical method that is used and followed in this report to be able to implement the gaze tracking system with the specified constraints. The theoretical method that is used in this report is a feature based gaze tracking approach, that uses three phases and utilizes facial recognition as well as image manipulation to get an estimation of the gaze vector. The estimated gaze vector is mapped to the screen with a second-degree polynomial transformation [3].

The development environment part of the method includes the choice of programming language, integrated development environment for compilation and debugging, as well as theoretically related choices for development.

The evaluation method is the final part of the method and is used to confirm that the implementation works. Further, this step is used to extract raw data from the gaze estimation for analysis and evaluation.

1.7 Delimitations

This thesis project's goal is, as previously mentioned, a real-time eye tracking system that runs on a regular computer and only makes use of a web camera. A simple web camera is desirable so that essentially anyone can use the system without any unusual, intrusive or expensive equipment. The system will be analyzed and evaluated by the precision of the subject's eye movements projected on the screen. The system will not in any particular way compensate for environmental differences, like light or factors rooted in the user's behavior such as head movement, since it requires a rather time-consuming and complex addition to the method. Additionally, it is assumed that only one user is visible to the camera during calibration and estimation so that no additional computation is made that will unnecessarily burden the system. Finally, the system will not account for the user wearing glasses, due to the glasses potentially producing a reflection that blocks the camera from properly identifying the eye features.

1.8 Outline

In this report, there are seven chapters.

Chapter one is an introduction to the thesis where the background is briefly explained and what problem will be researched and examined. Shortly the purpose, the goal, and the method are explained and also what connection it has to ethics and sustainability.

Chapter two is a recess of the theoretical background. The field in computer science called computer vision and its sub-field gaze tracking is explained. Further, the different categories of methods in gaze tracking and examples of them are presented.

Chapter three is the discussion and analysis of the methods chosen in this thesis to carry out the project. This includes the literature study, the method that is used for the gaze tracking itself as well as some choices related to the software development itself.

Chapter four is about the performed work. In this chapter, there is an explanation of the different modules that were implemented as well as in what order they were implemented. This chapter also explains how the evaluation of the system was done. I.e. what method was used to determine how accurate and precise the system is on an average computer.

Chapter five presents the final results of the thesis. It displays how the resulting system looks as a program and explains how it works. It also shows how the evaluation of the system went. The total average of the evaluations is displayed and explained.

Chapter six contains analyses about the results and discusses them. In other words, what the results mean and why they look the way that they do. Some ways the system could be changed to improve the accuracy and precision of it are also presented.

Chapter seven presents a conclusion of the thesis in relation to its purpose and goal, as well as potential future work that can be performed to improve the work.

Chapter 2

Background

This chapter will explain what computer vision and its sub-field gaze tracking are about. It will then delve into the different categories that methods of gaze tracking can be placed in, including a denotation of instances of said categories. The first type of category is based on how intrusive the hardware part of the system is for the user. The second type depends on the type of camera equipment used for tracking. Finally, an explanation will be made of how computer programs can work to recognize facial features either from still images or from a stream of video frames.

2.1 Computer vision

Computer vision is a field that, broadly speaking, concerns itself with the "methods for building artificial visual systems that interpret and organize visual information" [1]. In other words, computer vision is about keeping track of and interpreting the surroundings of the computer system. It can be used for a plethora of different applications such as assisting vehicles or robots to navigate their surroundings, building systems that aid the user in interpreting key elements of their surroundings, and building systems that track the gaze of the user; also known as gaze tracking.

2.1.1 Gaze tracking

A lot of research has been made on gaze tracking for a long time [4]–[14] with great variation in approaches to how the tracking is done. The variation can primarily be divided into categories based on the different equipment used. However, within each of these categories, there is a great number of approaches one can use to build a model for describing reality. These models are primarily differentiated between by their complexity since they all require different levels of mathematical ability to implement and consequently result in different ways for the computer program to perceive the eyes of the user.

2.2 Methods based on environment setup

There are two major setups of environment for gaze estimation methods, head-mounted environments and desktop environments. Methods that use head-mounted environments make use of some kind of hardware (often a camera and multiple IR-LEDs) which is attached to the user's head. The spatial relation between the eye and the camera being static, in combination with the reflection of the IR-LEDs, allows the system to get a consistent perception of the eye. This is in contrast to methods that use desktop environments, which may need complicated functions to extract the eye region before estimating the gaze. These functions are required since these methods are using cameras that are statically positioned and therefore need some way to estimate the relative position of the eye.

2.2.1 Head-mounted environments

Head-mounted environments methods make use of one or multiple cameras that are attached to the head. By attaching the hardware to the head, estimating the gaze becomes easier because one can achieve a clear and high-quality image of the eye directly, as opposed to searching the whole image just to extract the eye region. From this follows that there is less work spent on each frame to detect the eye and thus to estimate the gaze. It is also not necessary to consider significant head movements because the camera is attached to the head and moves accordingly. On the other hand, equipment that is used for head-mounted environments is sometimes rather expensive and hard to get a hold of. It is also not suitable for all areas of application because the equipment can be uncomfortable and hard to wear. Since it is essential for some of the said areas that the user experiences no hardware intrusion, an alternative is desirable.

2.2.2 Desktop environments

Methods that use desktop environments depend on one or multiple cameras that are fixed in position and thus do not need to be attached to the user, allowing for a non-intrusive tracking of the user's gaze. On the other hand, since the camera's position is fixed and the user's relation to the camera is innately unknown, the method needs to estimate the position of the user's eyes in some way.

One common way of solving this problem is to detect the user's face region. By doing this, the face region can be searched through to find and extract the eye region. Since the face region needs to be detected another problem arises, which is that of head movement. Either the method needs to assume that the user does not move the head at all during calibration and estimation, or the method needs to compensate for such movement in its gaze estimation in some way. From this follows that there are either higher demands placed on the user's interaction with the program or that a more complex implementation is needed, which is the trade-off compared to the expensive and intrusive equipment that head-mounted environments entail.

A lot of different types of cameras can be used. Web cameras, stereo cameras, zoom control cameras, multiple lens cameras, and control cameras are all possible options [15]–[17]. In "A low cost webcam based eye tracker for communicating through the eyes of young children with ASD" , Khonglah and Khosla [4] make use of a simple web camera and six IR-LEDs. In the study "Eye gaze tracking under natural head movements", Zhu and Ji [5] uses a camera and one IR-LED while in "Real Time Eye Gaze Tracking with 3D Deformable Eye-Face Model", Wang and Ji [6] only make use of a simple web camera. Using only a web camera is beneficial in situations when there is a need to be able to quickly set up the environment in different locations or when a low-cost hardware setup is desired. A typical desktop environment is displayed in Figure 2.1.

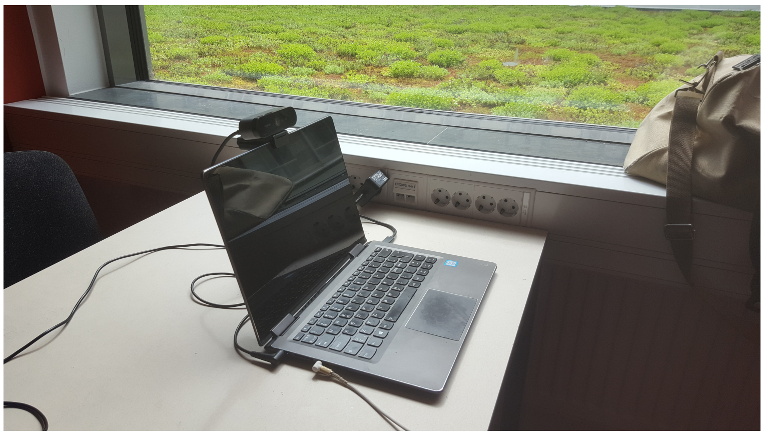


Figure 2.1: Desktop environment

2.3 Methods based on camera equipment

Gaze tracking estimation methods can be separated into two categories based on the type of camera equipment that they are using. The methods of the first category only make use of a single camera [3], [6]–[9], [18]–[20] while methods of the second category make use of more complex hardware systems and rather expensive equipment, such as multiple IR lights and cameras [10]–[15], [21].

2.3.1 Single camera

The single camera category is further divided into two sub-categories based on what type of camera is used. The types of cameras that exist are light-sensitive cameras and regular cameras. There are more studies done on regular cameras than there are on light-sensitive cameras since the need for gaze tracking with a regular camera is greater. This is primarily due to it being the significantly cheaper alternative of the two.

Light-sensitive camera

Methods that make use of a light-sensitive camera are efficient when fast real-time image processing is needed because their code implementations do not need to consider variations in environmental light, which means that less manipulation of image frames is required. Regardless, the accuracy is not so good, according to Francisco et al. [8] they managed to achieve an accuracy above 90% which is an average gaze error deviation of approximately 5 degrees, in "Detecting eye position and gaze from a single camera and 2 light sources", Morimoto et al. [19] the authors also managed to get a gaze estimation error deviation of 5 degrees.

Regular camera

Methods that only make use of a regular camera such as a simple web camera or a stereo camera are good in situations when the system needs to change location often since the setup is rather easy to move around. From this, it follows that the environment can differ and therefore various light conditions need to be accounted for by the method or similar light conditions must be ensured by the user(s) of the system. The manipulation of pixels to solve varying light conditions results in a more complex and heavier implementation, which in turn means worse performance. The accuracy of the gaze and the error in degrees varies relatively much, based on the method that is used. This is because the methods make use of different algorithms. "3D gaze estimation with a single camera without IR illumination" [7] achieved an error in the estimated gaze of less than 3 degrees, while the authors of "Eye Gaze tracking using an RGBD camera" [9] got a 4-degree gaze estimation error. In "Remote gaze estimation with a single camera" [18] the authors managed to get a gaze estimation error between 5 and 7 degrees. In "Real Time Eye Gaze Tracking with 3D Deformable Eye-Face Model" [6], which uses a web camera, the system had an average gaze estimation error of 3.5 degrees while in "Eye Gaze Tracking With a Web Camera in a Desktop Environment" [3], they managed to get a gaze error margin of 2.27 degrees.

2.3.2 Additional IR-cameras

Methods that make use of IR-cameras have rather accurate estimations of the gaze due to extra support from the IR-LED to produce a constraint of the corneal glints in the gaze tracking [11]. Depending on what method is used, type of cameras and how many IR-LEDs that are used, the accuracy varies. In "A Novel Approach to 3-D Gaze Tracking Using Stereo Cameras", Shih and Liu [15], "A robust 3D eye gaze tracking system using noise reduction", Chen et al. [11] and "Remote point-of-gaze estimation requiring a single-point calibration for applications with infants", Guestrin and Eizenman [13] all managed to get a gaze error estimation of <1 degree. While "Eye gaze tracking using an active stereo head", Beymer and Flickner [10] achieved a gaze point accuracy of 0.6 degrees but without head movement.

Anyhow, methods that make use of IR-cameras need to include different circumstances such as sunshine and reflections and also the position between the IR lights and where the camera is. This is because sunshine can affect the IR illumination if the camera is in direct contact with the light. If the subject is wearing glasses there is a possibility that the glass reflects the IR illumination. Therefore, the methods that use IR-cameras are limited to certain target groups and environments.

2.4 Recognition of facial features

As previously mentioned, in a desktop environment the eye region needs to be found and analyzed to estimate the gaze. Therefore the recognition of facial features is an important and useful step in tracking the gaze. Facial feature recognition can be done in several different ways.

Active appearance model [22] is one type of method which can be used for finding and recognizing facial features. It is a linear model of different object shapes and texture variation, which has been trained and learned from a set of training data. The training data is a set of face images combined with coordinates of landmark points. Active appearance model uses square techniques and the collected data to analyze and find the difference between the current estimation of appearance and the target image.

Active shape model [23] is related to the active appearance model but with a disadvantage that it only uses shape constraints together with information about the image structure near the landmark points. Therefore it does not use all the available information, i.e. all the textures across the target object. As a consequence of this active appearance model is slightly slower than active shape model but is more robust and accurate because more information in the image is used.

There is also the regression tree method [24] which computes a 2D object pose with a rough initial estimation [25]. The regression tree method also makes use of a training set but combines it with a regression function in the cascade [26]. The regression is learned by a gradient boosting [27] combined with an error loss function. Given a facial image and a face shape the regression updates the shape for each iteration until the face is aligned. The regression tree method is not just faster than active shape model but also more accurate for finding the facial features [24].

Chapter 3

Method

In this chapter, the different parts of the theoretical method are explained. First, the literature study is described and the most essential previous work is briefly explained. Then, a description of the method used for the gaze estimation itself is presented, split into three phases. This is followed by an explanation of the different choices that were made in relation to the development environment, i.e. what programming language was chosen and what other software was used to save time and to otherwise make the implementation easier. Finally, the evaluation method that is used to test the system is expounded.

3.1 Literature study

There are two major categories of gaze tracking estimation methods that make use of a 3D-based geometric eye model to estimate the gaze by mimicking the structure and the function of the human eye. The first method [11]–[15], [21] includes a complex hardware system and rather expensive equipment, such as IR lights. The second method [3], [6]–[9], [18]–[20] only includes a web camera to perform a 3D estimation of the gaze.

The method that is described in Section 3.2 is from the second category and was found in the literature study after going through all the material presented in the paragraph above, in Section 2.2, and Section 2.3. Before this method was chosen, a lot of different methods were gone through that were, for the most part, using one or more additional IR-cameras and was thus largely irrelevant to the desired implementation, since it is only supposed to use a single web camera in terms of hardware. These studies were however useful for understanding the general concept of gaze tracking and the limitations that exist for methods based on desktop environments using a single camera. The method that was finally chosen for gaze estimation is a method that is largely based on the one described in "Eye Gaze Tracking With a Web Camera in a Desktop Environment", Cheung and Peng [3]. That method is the same as the implemented one, with the exception that the implemented method does not estimate the head pose of the user. This is because head pose estimation requires some quite heavy mathematical implementation and would thus take too much time to implement for the scope of this project. A more detailed description of how the mapping function is formed in calibration and used in estimation, specifically the underlying math, can be found in a report written by Cherif et. al in "An adaptive calibration of an infrared light device used for gaze tracking" [28], was also found through this paper.

3.2 Gaze estimation

The method that is used is based on a three-phase feature based eye gaze tracking approach that makes use of the head position and eye features to get an accurate estimation of the gaze position in relation to the camera's position, originally created by Cheung and Peng in "Eye Gaze Tracking With a Web Camera in a Desktop Environment" [3]. The implementation in this project differs from the original method in the way that it ignores the head pose estimation, as mentioned in Section 1.7 since it is too time-consuming for this project. Because of this, the method is presented in a slightly different way. The first phase consists of detecting the eye region, the second phase deals with separating the features of the eye region to estimate the inner corner of the eye as well as the center of the pupil. In the third and final phase, these two reference points are used to calculate the gaze vector. It then uses the gaze vector as input to a mapping function that returns an estimated gaze point. Before the mapping function can be used, it must be formed in calibration. During calibration, the third phase is used to calculate the gaze vectors for each corresponding calibration point and establishing a common relationship between them.

A. Eye Region Detection

The first phase in the method is to detect the eye region which is done with a two-step method. The first step in this method is a face region detection which is done using a classic histogram of oriented gradients together with a linear classifier, an image pyramid, and a sliding window detection scheme [29]. The second step of the method is the facial landmark detection which is done with face alignment using an ensemble of regression trees [24]. A visualization of its result is illustrated in Figure 3.1.

A classic histogram of oriented gradients is used to utilize different lighting conditions. This is done by dividing the image into smaller spatial regions that can be called cells. Then a gradient vector is generated for each cell which is represented as a histogram. Each cell is then split into angular bins, where each bin corresponds to a gradient direction. This is to reduce the number of vectors and store the gradients' magnitudes. Furthermore, the cells are grouped into blocks together with their neighbors which are normalized, to ensure that it is invariant to illumination changes, i.e contrast and brightness. Finally, the histogram is combined with an image pyramid that scales up the image and combines it with a trained linear classifier [30] as well as a sliding window detection to scan across the image to detect object instances, in this case, the face region.

When the face region is detected and extracted from the image the eye region needs to be found and extracted. This is done using feature landmark detection[24]. The landmark detection uses a cascade of regressors which iterates until the estimated shape fits the expected shape. The cascade predicts an update vector from the image which is added to the current shape estimation in each step. This process is described by Equation (3.1) below.

$$S^{t+1} = S^t + r_t(I, S^t) \tag{3.1}$$

Where I is the face region image, S = $(X_1^T, X_2^T, \dots, X_n^T)$ is the shape vector and denotes the coordinates of all the n landmark points in the image I. S^t is the current estimation of S and r_t is each regressor step. This is further combined with a trained shaped prediction data set for each iteration but going into details how that works is outside the scope of this report.

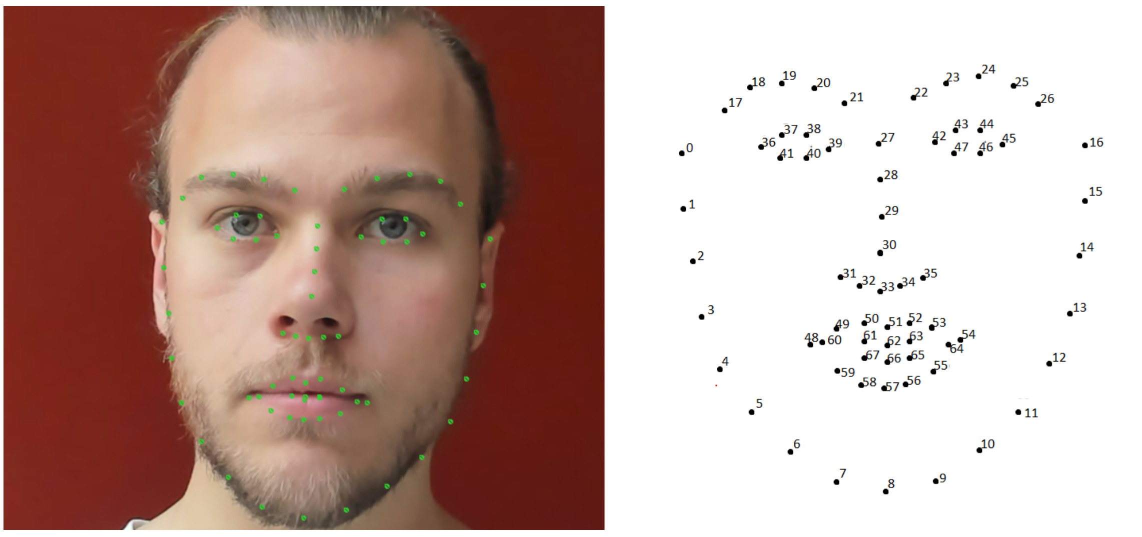


Figure 3.1: Landmark detection

For the reader that wants to know more about Eye Region Detection and the displayed method, a further understanding can be achieved by reading the following reports. "One millisecond face alignment with an ensemble of regression trees", Kazemi and Sullivan [24], "Cascaded Pose Regression", Dollár et al. [26], and "Face Alignment by Explicit Shape Regression" Cao et al. [25].

B. Eye Features Detection

Landmark points are attained from the face detection according to the model of Figure 3.1 to get the pixel coordinates of the points around the eyes. The landmark points include the inner corner of the eyes which is one of the parameters required to estimate the gaze vector. Since it is the second point that the gaze vector consists of, the center of the pupil must also be extracted, so that the vector can be calculated. The center of the pupil is not one of the landmark points and must thus be estimated in another way. The image manipulation of the eye region that aims to achieve this is illustrated in Figure 3.2 and comes from "Automatic pupil detection and extraction", Gupta [31].

First, the eye region image is inverted, then converted to grayscale. Finally, a binary threshold filter is applied to it. The binary threshold filter results in a purely black and white image and the final step is to distinguish the collections of connected pixels that are white, called blobs. After the image manipulation has been performed, a blob detection function is used for finding the largest blob in the eye region. Given the order of filters to the eye region, this blob should be the pupil. The center of this blob is then used as the pupil center.

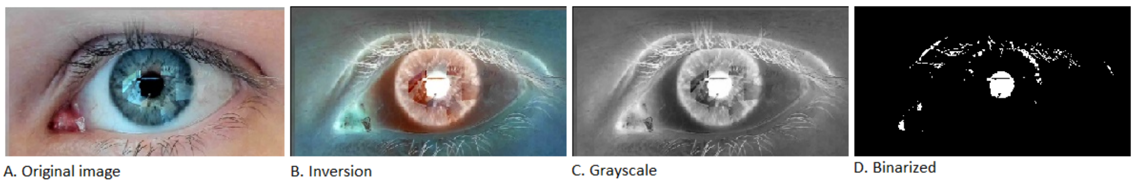


Figure 3.2: The changes in the eye region through the four stages of image manipulation, to prepare it for pupil detection.

C. Gaze Vector and Calibration

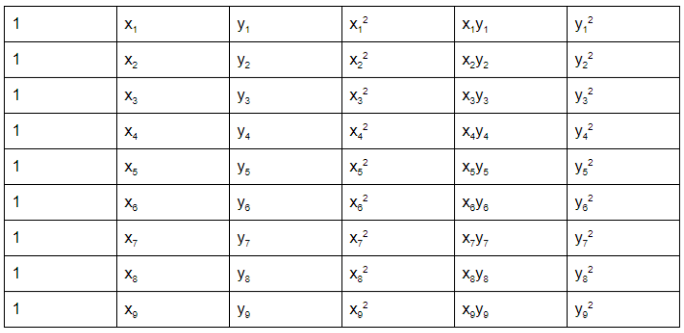
To extract the estimated point on the screen, an estimated gaze vector needs to be calculated first. The gaze vector is defined by the simple equation

$$g = p_{corner} - p_{pupil} \tag{3.2}$$

where p_{corner} is the inner eye corner and p_{pupil} is the center of the pupil. This becomes a nearly trivial calculation at this point since both of these points are already obtained from the previous phases. Before the vector g from Equation (3.2) can be used to estimate points on the screen, a mapping function must be formed in a calibration procedure. The calibration procedure provides a set of nine different points on the screen. The number of calibration points is chosen partially to allow for an even spread along the horizontal and vertical axis (3x3), and partially to be greater than the number of coefficients in the mapping function described below, which is required for the function to work [28]. One calibration point at a time is presented on the screen and an estimated gaze vector is recorded when the next point is stepped to.

The mapping function that is used is the same mapping function used in "Eye Gaze Tracking With a Web Camera in a Desktop Environment", Cheung and Peng [3], and is a second-degree polynomial transformation that is derived from the method described in "An adaptive calibration of an infrared light device used for gaze tracking", written by Cherif et. al [28]. The mapping function can be described as follows.

For each calibration point, the corresponding estimated gaze vector is added as a row in the matrix M, a matrix with nine rows and six columns, see figure 3.3.



1	x_1	y_1	x_1^2	x_1y_1	y_1^2
1	x_2	y_2	x_2^2	x_2y_2	y_2^2
1	x_3	y_3	x_3^2	x_3y_3	y_3^2
1	x_4	y_4	x_4^2	x_4y_4	y_4^2
1	x_5	y_5	x_5^2	x_5y_5	y_5^2
1	x_6	y_6	x_6^2	x_6y_6	y_6^2
1	x_7	y_7	x_7^2	x_7y_7	y_7^2
1	x_8	y_8	x_8^2	x_8y_8	y_8^2
1	x_9	y_9	x_9^2	x_9y_9	y_9^2

Figure 3.3: Matrix M, containing the transformed and combined coordinates of the estimated gaze vectors from the calibration.

After the matrix has been built, its inverse is found with Equation (3.3).

$$M^{-1} = (M^T M)^{-1} M^T \tag{3.3}$$

M^T is the transpose matrix of matrix M. Multiplying M with its transpose results in a matrix with an equal amount of rows and columns, which makes it possible to take the inverse of it. Finally the transpose of M is multiplied back into it to get a matrix with the inverted size of M. The coefficient vectors can then be obtained from Equations (3.4) and (3.5).

$$a = M^{-1} X_s \tag{3.4}$$

$$b = M^{-1} Y_s \tag{3.5}$$

where X_s and Y_s are 9x1 vectors containing all the x and y coordinates of the known calibration points on the screen. This entire process is essentially shorthand for applying the least squares method to minimize the correction between the gaze vectors and the calibration points. Estimating a gaze point based on a gaze vector can then finally be achieved with Equations (3.6) and (3.7).

$$S_x = a_0 + a_1 v_x + a_2 v_y + a_3 v_x^2 + a_4 v_x v_y + a_5 v_y^2 \tag{3.6}$$

$$S_y = b_0 + b_1 v_x + b_2 v_y + b_3 v_x^2 + b_4 v_x v_y + b_5 v_y^2 \tag{3.7}$$

Here, (S_x, S_y) are the estimated coordinates on the screen, (v_x,v_y) is the gaze vector, and (a_1, ..., a_5) and (b_1, ... , b_5) are the coefficient vectors from the mapping function found in Equations (3.4) and (3.5).

3.3 Development environment

The chosen programming language is C# and Visual Studio is the chosen IDE. C# comes with the benefits of being an object-oriented language which provides a clear modular structure of the program, makes it easier to maintain, and allows for faster development because of rich libraries [32]. At the same time, using C# allows for the use of ported libraries from C and C++.

The use of Visual studio together with C# makes it easy for the developer to start building an application without thinking about the graphical user interface that much since Visual Studio provides some quick and practical tools to build it - a great foundation to build and test a system prototype [33]. Moreover, there are many different libraries that are supported in C# for instance for image handling, face detection, and eye region detection. These facilitate the whole implementation of handling image streams from the web camera, finding the eye region and estimating the gaze vector.

Git is a tool for version control and it also provides the ability to share code between developers through services like GitHub. Therefore a repository on GitHub has been used for this project, making it easy to work on different parts of the implementation at the same time.

3.4 Evaluation method

The evaluation part of the method exists to establish that the implementation works and further, to be able to extract the raw data from the gaze estimation so that it can be compared to the point where the test subjects supposedly are looking in reality. This is done to investigate and analyze how accurate the system is.

The testing setup and environment should be the same for each subject so that the conditions are the same for everyone. Therefore should all the testings be performed in a rather close period of time due to variance in brightness. The distance between each subject and the screen should be approximately 45 cm.

The choice of subjects should be based on a variance in their eye color and eye shape. The testing phase happens according to the following steps. 1. Each subject starts by looking at the displaying video stream to be able to place themself in the middle of the image. 2. A calibration stage is performed together with a supervisor. This is done in the following way, 9 different points are shown at the screen, one at a time. The subject should then carefully look at each point while keeping their head steady until the process is done. 3. An estimation stage, each subject looks at nine new points and performs a similar procedure as in the calibration stage. These nine points can then be recorded and written to a file along with the estimations for further evaluation.

Chapter 4

Implementation and testing

Chapter 4 starts, with Section 4.1, by breaking down the implementation process and how the desired system is divided into different modules, to get a clearer understanding of the order of implementation that the system requires. Then, the implementation of each module is described, including the different libraries and data structures that are used. In the second section of this chapter, Section 4.2, the work that was performed to test and evaluate the system is explained. The circumstances of the testing are laid out and the testing process is explained.

4.1 Implementation

The implementation of the system is divided into five modules. Gaze estimation, mapping function, calibration, feature detection, and image handling. The relationship between these is described in Figure 4.1 by the arrows that point from each module to its respective dependencies. The goal with the implementation is to implement a live gaze tracking/estimation system. To be able to implement live gaze estimation there is a requirement for being able to find an estimated gaze vector as well as there being a mapping function that projects the vector on the screen. To accomplish this, the system must be able to detect facial features so the gaze vector can be calculated. It also depends on forming the mapping function that describes the relationship between the user's estimated gaze vector and points on the screen. In order for this mapping function to be formed, it needs to be calibrated with known points on the screen combined with their respective estimated gaze vector. For the calibration to work, it needs to generate the points on the screen but also needs the gaze vector estimation (feature detection module) to be implemented. In other words, the gaze estimation, the mapping function, and the calibration are all dependent on the module of feature detection. To be able to implement the feature detection, the program needs to be able to handle images from the web camera's video stream as individual image frames. The part that all other parts of the program depend on is thus the image handling, which consequently has to be implemented first.

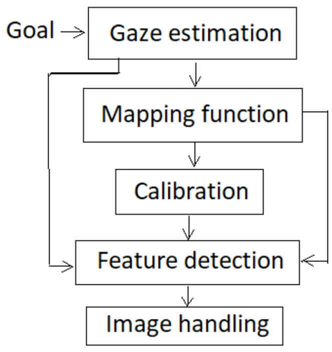


Figure 4.1: Topological implementation order. An arrow points at a dependency of the module it starts from.

A. Image handling

By using the AForge library [34], all camera devices connected to the computer as well as their supported modes (resolution, frame rate, and color depth) can be found and selected between. The library provides a frame event handler that is run every time the program receives a new frame from the camera, which happens at a frequency according to the selected frame rate. In the handler, an image frame from the camera is represented as a Bitmap object which is used to read from and write to individual pixels. The Bitmap object is cropped to half of its width and half of its height since the face is the relevant part and is typically positioned around the center of the image. By doing this, the number of pixels searched through becomes substantially reduced and so the processing time too. This cropping is performed using a filter function from the AForge library. Further, the Bitmap object is used in the feature detection function to extract an estimated gaze vector.

B. Feature detection

Feature detection is the part of the program where the gaze is estimated. To estimate the gaze, the face region and consecutively the eye region need to be extracted first. Both of these things can be accomplished with the DlibDotNet library [35]. The DlibDotNet library handles all its image data with its own Matrix object. Since the AForge library uses a Bitmap object, a conversion between the Bitmap object and the Matrix object is necessary and is thus the first thing to implement in feature detection. DlibDotNet face detection makes use of the regression tree shape prediction method described in Section 2.4. It uses training data to match the shape of the face and the shape model it uses is illustrated in Figure 3.1. In this implementation, the training data that is used is obtained from the publisher of DlibDotNet through a link in the source code. By using this data, time is not wasted on making an own set of training data that most likely would not be as accurate as the one that the publisher provides.

Landmark points are attained from the face detection according to the model of Figure 3.1 to get the pixel coordinates of the points around the eyes. The landmark points include the inner corner of the eyes which is one of the parameters required to obtain the estimated gaze vector. Since it is the second point that the gaze vector consists of, the center of the pupil must also be extracted, so that the vector can be calculated. The manipulation of the eye region is illustrated in Figure 3.2. First, the eye region image is inverted, then converted to grayscale. Finally, a binary threshold filter is applied to it. All of this image manipulation is done with AForge library filter functions applied to the Bitmap object, based on pixel locations of the landmark points, found in the DlibDotNet Matrix. The binary threshold filter results in a pure black and white image and the final step is to distinguish the collections of connected pixels that are white, called blobs. After the image manipulation has been performed, a function from the AForge library is run to find the largest blob in the eye region. Given the order of filters to the eye region, this blob should be the pupil. The center of this blob is then used as the pupil center. With the coordinates of the inner eye corner as well as the pupil center, the estimated gaze vector is calculated to be used in the calibration and gaze estimation modules.

C. Calibration

The calibration is the module that forms the mapping function, which in conjunction with the feature detection is what allows the live gaze estimation to be done. The calibration produces nine points evenly spread horizontally and vertically over the screen. After the points have been produced, they are kept in a list that is shuffled so that the order they are displayed in is random. When a calibration point is displayed and the program moves to the next point, the most recent estimated gaze vector is added as a row to the matrix M, as described in Section 3.2. M is represented using a Matrix object from the library Math.NET Numerics [36]. When all the nine points have been shown and the last one is removed from the screen, all the rows of the matrix have been filled with gaze vector data. At this point, the Math.NET Numerics library is used to perform the matrix operations described in Equation (3.3). The resulting inverse matrix of M is then used in matrix multiplication respectively with the matrices that hold the x- and y-coordinates of the calibration points on the screen. The result of the multiplication is the coefficient vectors a (Equation (3.4)) and b (Equation (3.5)).

D. Mapping Function

The mapping function translates an estimated gaze vector to a point on the screen by multiplying it with the coefficient vectors a and b. The function takes a gaze vector with an x- and y-component which is multiplied with the coefficient vectors and added as shown in Equations (3.6) and (3.7). The estimated gaze point on the screen is obtained by combining the results from the two multiplications, each representing one of the two coordinates.

E. Gaze tracking

Gaze tracking is the part of the system where the actual translation to the screen happens. When all of the above modules are implemented, the gaze tracking module can use the live feature detection to get an estimated gaze vector from each image frame provided by the image handling module. The vector is used as the argument to the mapping function which returns a point on the screen. The estimated gaze point is displayed on the screen and the gaze tracking is done.

4.2 System evaluation

The evaluation of the system includes two hardware pieces, a web camera for producing the image stream and a computer to run the system. The camera that is used is a Logitech Brio web camera with a resolution of 2560 x 1440 pixels and a frame rate of 30 fps. The camera is installed on top of the monitor of the computer and at its horizontal center. The computer used is a laptop with the following specifications:

- **Monitor:** 14" 16:9, 1920 x 1080 pixels
- **Operating system:** Windows 10 Home 64-bit
- **CPU:** Intel®Core$^{\text{TM}}$ i7-7500U @2.70GHz
- **RAM:** 8GB DDR4 2133MHz
- **Graphics:** Intel HD Graphics 620

The same testing location is used for all subjects within one hour, during which the light conditions are roughly the same. The position in the room is also the same for everyone. The distance between each subject and the camera is approximately 45cm.

There are six individual subjects. They were chosen based on their availability but a variance in eye color, eye shape, and the visibility of the subject's eyes was still maintained. The tests were performed according to the following process.

Each subject starts by looking at the displayed video stream from the web camera to make sure that their face is around the center of the image. When they are sitting in the right position, approved by a test supervisor, the calibration stage is started.

Starting from the calibration stage, each subject tries to hold their head more or less completely still. During calibration, one calibration point shows up at a time and the subject signals when they start to look at it. The test supervisor waits a short moment to make sure the camera is not lagging behind or at the very least catches up, then presses a button which saves the estimated gaze vector of the subject and moves on to the next point. After the calibration is done, the estimation can start since the mapping function has been formed.

In the estimation stage, each subject looks at the same nine points as in calibration, one point at a time. When the subject signals that their gaze is locked on to the point, two seconds pass before the test supervisor presses a button that writes the 15 most recent gaze point estimations to a file. The x- and y- coordinates of the actual point on the screen as well as of the estimated gaze point were written to the file, with each datum separated by a comma. Each estimation was separated by a line break. When all the nine points have had their 15 estimated gaze points written to the file, the test is complete for that test subject. The raw data in the file is opened as comma-separated values in Google Sheets where each column gets to represent either the x- or the y-coordinate of the actual and estimated points respectively.

The results are further presented in Section 5.2.

Chapter 5

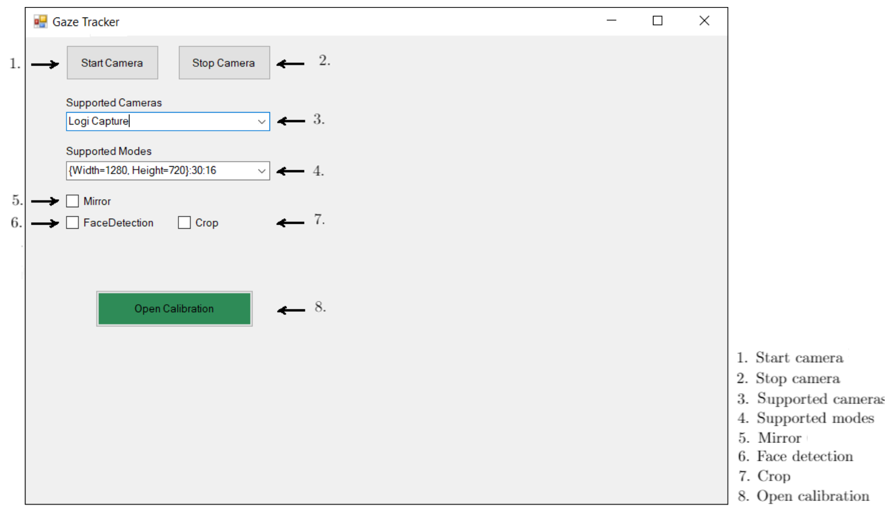
Result

This chapter consists of two sections. The first section presents the prototype that resulted from the project work, describing its different parts and how to use it. The second section shows the results from the system evaluation, as described by Section 3.4 and 4.2.

5.1 Final prototype

The system itself of the final prototype consists of four parts. The start menu, the calibration menu, the calibration state, and the gaze estimation.

The first part is the start menu. It is demonstrated in Figure 5.1 and consists of four different functions. Number 1 and 2 in Figure 5.1 are the start and stop camera functions. Their purpose is very straight-forward, they start and stop the camera and mode that is currently selected. Number 3 in Figure 5.1 is the "Supported Cameras" function which lists all different cameras that are connected to and supported by the system. Number 4 in Figure 5.1 is "Supported Modes" and lists all the different settings available for the chosen camera. Its numbers are, in order of occurrence, resolution, frame rate (frames per second), and color depth (number of bits). Number 5 in Figure 5.1 is a mirror function that simply mirrors the image stream, i.e. reflects it horizontally. Number 6 in Figure 5.1 is "Face detection" which starts the process of detecting the face, landmark detection, eye region detection and then extracting the gaze vector. This box must be checked for the calibration and gaze tracking to work. Number 7 in Figure 5.1 is a crop function that crops the image stream to half of its width and half of its height, in order to reduce the processing time spent on each frame. Number 8 in Figure 5.1 is the "Open calibration" function which opens a new window for the calibration procedure.



Supported Cameras: Logi Capture

Supported Modes: {Width=1280, Height=720}:30:16

Figure 5.1: Start Menu. Start and stop camera, choice of camera, resolution, fps, color depth, filters, face detection, and calibration alternatives.

The "Calibration Menu" section which is not demonstrated in any figure only has two options. One is "Start calibration" which initiates the calibration process. The other, "Exit calibration", leads back to the start menu in Figure 5.1.

The "Calibration State" which is demonstrated in Figure 5.2 is when the calibration happens and should be used as follows. Upon pressing the spacebar button, it starts by showing one out of the nine calibration points chosen randomly, which can be seen in Figure 5.2. When the user is looking at the point, the user (or a supervisor) should press the spacebar button to continue to the next calibration step. The program will then randomly select a new point out of the remaining ones and show it on the screen. The process is repeated until the user's eyes have been calibrated with respect to all nine points. Pressing the escape button at any point during calibration will prompt the user with a dialog box that asks if the user wants to cancel the calibration, warning them that any data recorded will be lost. If the user selects 'Yes', the calibration ends and returns to the "Calibration Menu" state. If the user selects 'No', the calibration resumes as if nothing happened. When the calibration has stepped through all the nine calibration points the program automatically enters the "Gaze Estimation" state as displayed in Figure 5.3 and described below.

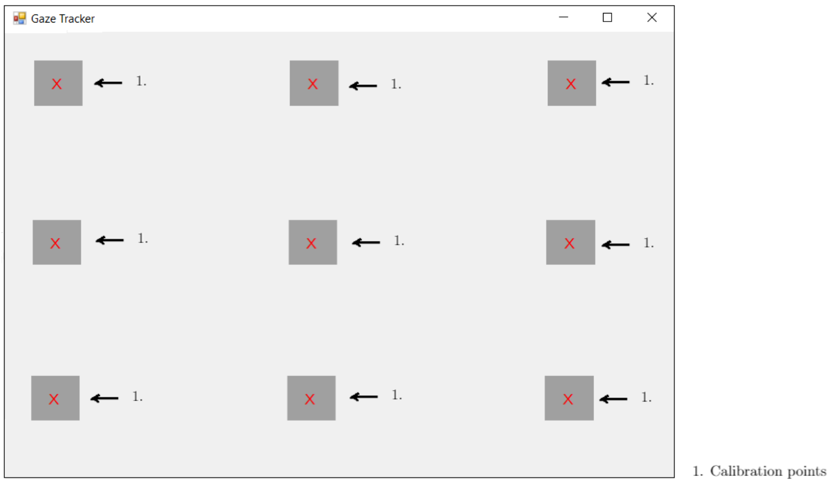


Figure 5.2: Calibration State.

The final part of the system is the "Gaze Estimation" state, which is demonstrated in Figure 5.3. The figure illustrates the real-time estimation of the user's gaze, which is projected onto the screen and displayed as a green square, see Number 1 in Figure 5.3.

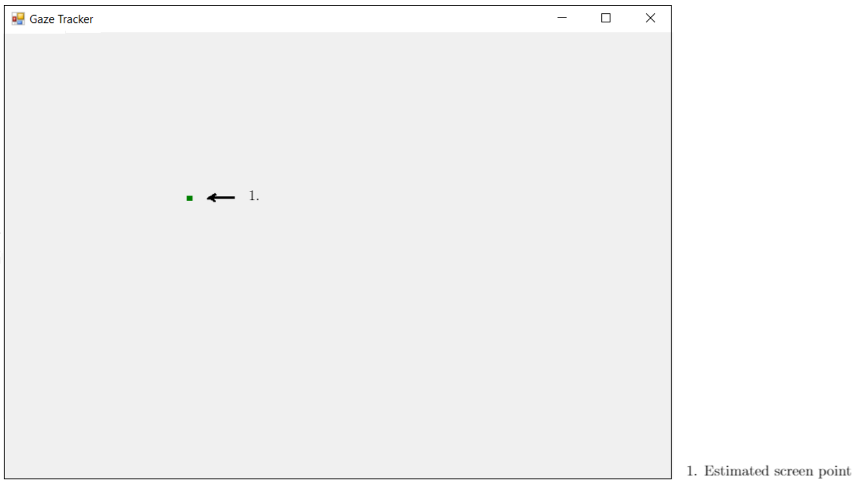


Figure 5.3: Gaze Estimation

5.2 Estimation evaluation

Below, the results of the estimation tests that were described in Section 4.2 are presented.

The results contain 810 data points, spread on six different test subjects that each partook in one calibration and one estimation session each, for a total of 135 estimations per test subject (15 estimations per test point with nine test points per estimation session).

The raw data of each subject is used to calculate the difference between the actual points and the estimated points in the horizontal and vertical directions, as described by Equations (5.1) and (5.2) respectively.

$$diff_x = actual_x - est_x \tag{5.1}$$

$$diff_y = actual_y - est_y \tag{5.2}$$

Further, the data resulting from these two equations describe how far away the system's estimations are from the actual gaze points in the two horizontal and the two vertical directions respectively. A negative value means for the x-coordinate that it is estimated too far to the left and for the y-coordinate that it is estimated too far down. A positive value then obviously means for the x-coordinate that it is estimated too far to the right and for the y-coordinate that it is estimated too far up.

The resulting data is plotted in scatter diagrams where the center represents the actual screen point and the blue data points represent one estimation. However, 149 (around 18.4%) of the 810 data points are excluded from the diagrams because the estimations are more than 2000 pixels away from the screen point either in the horizontal, the vertical, or both directions. The diagram for each test subject as well as the one with the combined data can be found in Appendix A.

In Figure 5.4 the estimated gaze points of all the test subjects combined are plotted in relation to the center. The center of the diagram represents the actual position of the known screen point during the tests. The plotted points represent the relative position of the estimated points to the point the subjects were supposedly looking at.

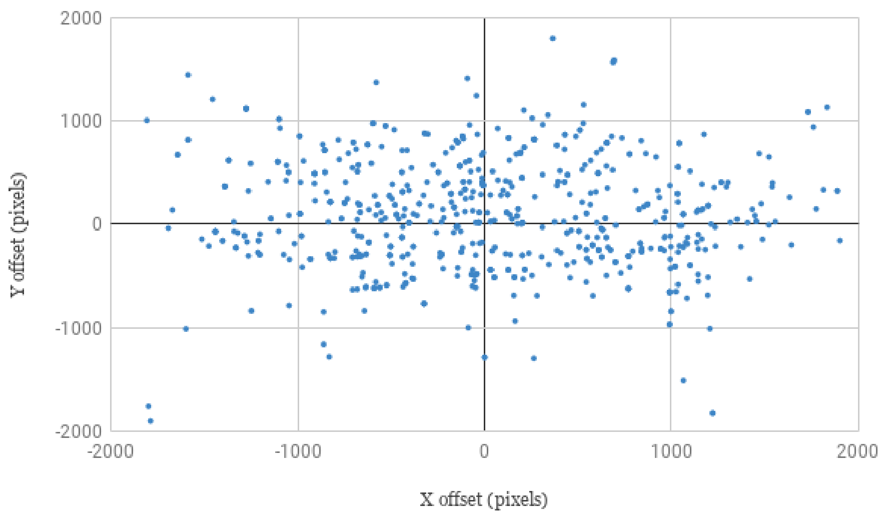


Figure 5.4: The offset in pixels for all the subjects' estimated gaze points in the x- and y-direction compared to the actual screen point.

This figure provides a decent overview of how the spread of estimations looks like, among the points that are within 2000 pixels of the center in both directions. However, to get a more concrete and quantitative understanding of how far the estimations are from their respective point on the screen, the distance for each point (including the ones that were excluded in Figure 5.4) is calculated, according to Equation (5.3).

$$distance = \sqrt{(diff_x)^2 + (diff_y)^2} \tag{5.3}$$

The resulting data are grouped into subsets of data in intervals based on powers of 2, starting at 0 to 2^5 and ending at 2^{15}. These ranges are chosen so that there is a reasonably proportionate division of frequencies among the ranges. These subsets of data allow for plotting a histogram where the y-axis describes the number of data points that fall within a certain range of distances and the x-axis represents all these different ranges. The resulting histogram is shown in Figure 5.5. Each column represents the number of distances that are greater than the number on their left and less than or equal to the number below them. E.g. the tallest column represents the number of points that were at a distance greater than 512 pixels but less than or equal to 1024.

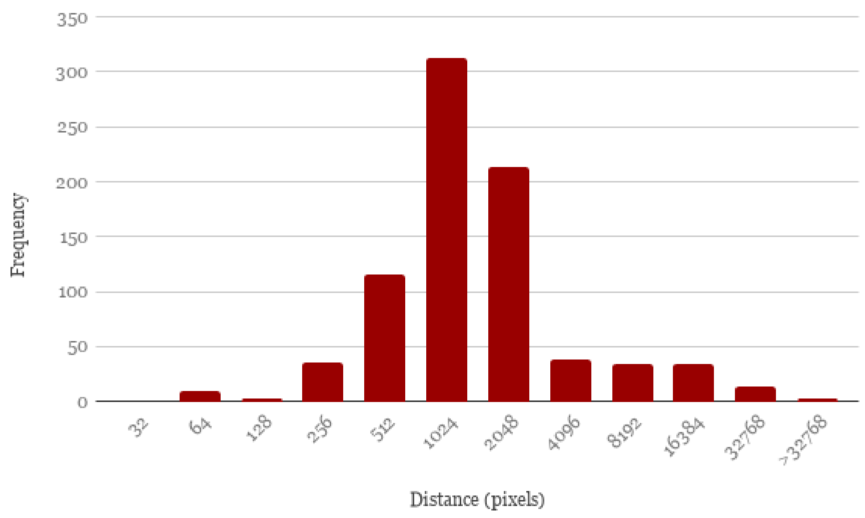


Figure 5.5: A histogram describing the distribution of distances (pixels) among the estimations from their respective points on the screen that they are supposed to be estimating.

Finally, the mean and median are calculated for Equations (5.1) and (5.2). The resulting means give a difference of approximately 262 pixels for the x-coordinates and -487 pixels for the y-coordinates. The resulting medians end up being 33 pixels and 52.5 pixels respectively.

Chapter 6

Discussion and analysis

Chapter 6 consists of two major sections that provide an analysis of the results presented in Section 5.2 and respectively, a discussion about the different parts of the project as well as how they may have impacted the results.

6.1 Analysis of results

Not much analysis of the results is required to realize that the system is working extremely poorly. While Figure 5.4 may give the impression that the offset generally is not that awful, one must consider the excluded data points, as well as the points that spread along the edges of the scatter diagram since they are also greatly imprecise estimations. Figure 5.5 gives a much clearer picture of how poorly the system is performing. Several hundred data points (around 40% of all estimations to be more exact) end up in the intervals that are > 1024 pixels, meaning the estimations are very likely to be outside the screen area since the maximum distance for a 1920x1080 resolution screen from corner to corner is around 2200 pixels. Many points are even tens of thousands of pixels away from the actual point, meaning they are several screens worth of space away from their target.

Finally, a look is taken at the respective mean of all the data that resulted from Equations (5.1) and (5.2), briefly presented at the end of Section 5.2. Although these means suggest a slight bias towards estimating above and to the right of the actual point, the corresponding medians tell us a different story. If there was indeed an estimation bias, the medians ought to be of similar magnitude and the same direction as their respective means. This is however not true since the median of the differences in the y-direction is positive as opposed to the negative mean. Additionally, the mean and median of the differences in the x-direction suggests that there is none or very little bias. This is because the discrepancy between them is rather significant but more importantly, the median is relatively close to zero.

6.2 Discussion

In this section, the impact on results will be discussed from a perspective of the gaze tracking method described in Section 3.2, the implementation described in Section 4.1, and respectively the evaluation performed as explained in Section 4.2. Possible solutions to the problems of these impacts are also suggested.

6.2.1 Method

The method that is used in this report is missing a rather big section to be able to estimate the gaze relatively accurately. That section is the estimation of a head pose. This means that there is no compensation for head movement in this project which affects the gaze estimation and causes gaze vector errors. Under the circumstances that were used during the evaluation test, a potential source of errors was found. The estimated gaze is given from the pupil center and the inner eye corner, i.e $g = p_{corner} - p_{pupil}$. If there is a pixel deviation in the estimated gaze vector of only one pixel, there is approximately a 120-pixel deviation on the screen.

To include head pose estimation, the AWPOSIT algorithm [3] or the POSIT algorithm [37] can be included in the method. Both algorithms utilize the idea of using an estimation of the head position, such as the facial features recognition described in Section 2.4, to be able to calculate the head movement, which in turn can be used to compensate for the error in the estimated gaze vector.

Furthermore, there is also a huge error that can occur due to the pupil detection. The logic seems to work in theory, but does not hold in reality. This is because the theoretical reasoning behind the pupil detection does not consider differences in environment and eye color. For instance, for very dark eyes it will be harder to distinguish the iris and the pupil in an environment with little light. In contrast, if too much light is present in the environment, a reflection might occur in the eye that obscures the pupil and consequently makes the blob detection select something other than the pupil. Because of these reasons, the estimation of the pupil center went wrong in some of the cases. Since the eye region, given the test conditions that were used for the evaluation, is typically around 100 pixels wide and 30 pixels high, a faulty pupil detection can result in an error in the gaze vector of up to around 100 pixels. This would then, in turn, result in a massive pixel deviation on the screen since it would estimate the user's gaze to be in a completely different direction from reality. Additionally, because the screen only makes up a part of the user's field of view, detecting the pupil at the edge of the eye region would mean the user is looking far outside of the screen region, which typically should not be happening neither during calibration nor during estimation.

The choice of evaluation method was inspired by the evaluation method described in "Eye Gaze Tracking With a Web Camera in a Desktop Environment", Cheung and Peng [3]. There were some changes in the method due to limitations in terms of time and access to a large pool of test subjects. The major change that was made was the number of testing iterations that were performed on each test subject. Hence, there is a larger risk for errors compared with using larger sets of testing data. Another thing to consider is that the reference points on the screen, that were used in the evaluation, were the same as the ones used in the calibration stage. From this, it follows that the precision might seem better than it is since the mapping function most likely works better around the points it was calibrated with than elsewhere on the screen.

6.2.2 Implementation

There exists a problem in the implementation of the pupil detection. This is because blinking is not considered in the implementation. When the user blinks, the pupil cannot be found because it is partially or completely obstructed by the eyelid. This results in a gaze vector that is completely wrong. This is because the pupil detection chooses a point from the edge of the eye region if no pupil can be found. This results in a large deviation in the estimated gaze vector which in turn causes an even larger pixel deviation on the screen. This problem could be remedied by discarding frames where either the eye region is too small, or the gaze vector is too different from the gaze vectors found in calibration.

The calibration implementation could also be done differently. Instead of using gaze vector data from one frame per calibration point, a median value out of, e.g. 15 frames, could result in a more accurate and precise calibration. This is because several factors that can cause the gaze vector in the calibration stage to be skewed, blinking included, thus causing the entire mapping function to be practically useless. By taking the median vector of some number of samples, one or a few outliers among the vectors do not affect the total calibration. This allows the user to act more natural in the calibration and estimation stages since inevitable behavior like blinking and brief straying of the gaze would not affect the end result.

6.2.3 Evaluation

The testing was done using a Logitech Brio web camera with a resolution of 2560 x 1440 pixels and a frame rate of 30 fps. This was too demanding for the computer that was used and consequently caused a delay in the program's image stream which forced the test subjects to wait for the camera to catch up under the entire calibration and estimation processes. From this follows that there might be two reasons for error in how the calibration was done and how the estimation data were extracted for each subject. The first reason is that the subject had to stare at each point over a period of time to counter-act the camera falling behind. This could lead to the gaze drifting away, due to the difficulty of focusing one's gaze on a single point for extended periods of time. Second, because of the camera delay and despite the efforts to counter-act it, there is no guarantee that the subject's gaze is directed at the correct spot on the screen when the gaze vector is recorded in the calibration and estimation processes.

Chapter 7

Conclusion and future work

7.1 Conclusion

While there is a lot of research and practical software libraries that can be used as a foundation for the implementation of gaze tracking systems, there are many things that can go wrong in the way it is implemented, tested as well as with the method that is used. Gaze tracking using only a web camera requires an incredibly well thought out method and implementation thereof since it has no natural way of handling e.g. head movements and changes in light conditions. One can attempt to neutralize these problems by giving certain instructions to the user during testing, but without the software to help solve these problems it is almost impossible to get an accurate and precise estimate of the user's gaze.

7.2 Future work

In this implementation, there is a lack of accuracy primarily because of two identified reasons. Partially because the pupil detection is unreliable and partially because there is no head pose compensation. Therefore, a good idea for future work to improve the system would be to implement a head pose compensation algorithm, e.g. the AWPOSIT algorithm [3]. Another idea for improvement would be to find a better method for pupil detection. One possible such method would be the one used in "Eye Gaze Tracking With a Web Camera in a Desktop Environment" [3], which is based on a special type of histogram equalization called locality sensitive histograms. Additionally, one might want to change how the calibration is done to ensure that it is more error tolerant, for instance by using the median of a gaze vector sample, as suggested in Section 6.2.2.

References

[1] A. Kadambi, A. Bhandari, and R. Raskar, *Computer Vision and Machine Learning with RGB-D Sensors*. 2014, pp. 3–26, ISBN: 978-3-319-08650-7. DOI: 10.1007/978-3-319-08651-4. [Online]. Available: https://www.kth.se/rpl/research/computer-vision-and-machine-learning-1.680751.

[2] Lexplore AB, *About Us*, 2019. [Online]. Available: https://www.lexplore.com/about-us/ (visited on 05/23/2019).

[3] Y. M. Cheung and Q. Peng, "Eye Gaze Tracking with a Web Camera in a Desktop Environment", *IEEE Transactions on Human-Machine Systems*, vol. 45, no. 4, pp. 419–430, Aug. 2015, ISSN: 21682291. DOI: 10.1109/THMS.2015.2400442. [Online]. Available: http://ieeexplore.ieee.org/document/7055334/.

[4] J. R. Khonglah and A. Khosla, "A low cost webcam based eye tracker for communicating through the eyes of young children with ASD", in *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, IEEE, Sep. 2015, pp. 925–928, ISBN: 978-1-4673-6809-4. DOI: 10.1109/NGCT.2015.7375255. [Online]. Available: http://ieeexplore.ieee.org/document/7375255/.

[5] Zhiwei Zhu and Qiang Ji, "Eye Gaze Tracking under Natural Head Movements", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, IEEE, pp. 918–923, ISBN: 0-7695-2372-2. DOI: 10.1109/CVPR.2005.148. [Online]. Available: http://ieeexplore.ieee.org/document/1467364/.

[6] K. Wang and Q. Ji, "Real Time Eye Gaze Tracking with 3D Deformable Eye-Face Model", in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, IEEE, Oct. 2017, pp. 1003–1011, ISBN: 9781538610329. DOI: 10.1109/ICCV.2017.114. [Online]. Available: http://ieeexplore.ieee.org/document/8237376/.

[7] J. Chen and Q. Ji, "3D gaze estimation with a single camera without IR illumination", in *2008 19th International Conference on Pattern Recognition*, IEEE, Dec. 2009, pp. 1–4, ISBN: 978-1-4244-2174-9. DOI: 10.1109/icpr.2008.4761343. [Online]. Available: http://ieeexplore.ieee.org/document/4761343/.

[8] F. Vicente, Z. Huang, X. Xiong, F. De La Torre, W. Zhang, and D. Levi, "Driver Gaze Tracking and Eyes off the Road Detection System", *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 2014–2027, Aug. 2015, ISSN: 15249050. DOI: 10.1109/TITS.2015.2396031. [Online]. Available: http://ieeexplore.ieee.org/document/7053946/.

[9] X. Xiong, Q. Cai, Z. Liu, and Z. Zhang, "Eye gaze tracking using an RGBD camera", in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing Adjunct Publication - UbiComp '14 Adjunct*, New York, New York, USA: ACM Press, 2014, pp. 1113–1121, ISBN: 9781450330473. DOI: 10.1145/2638728.2641694. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2638728.2641694.

[10] D. Beymer and M. Flickner, "Eye gaze tracking using an active stereo head", 2003, pp. II–451–8, ISBN: 0-7695-1900-8. DOI: 10.1109/cvpr.2003.1211502. [Online]. Available: https://pdfs.semanticscholar.org/fdd1/41a4fffc490fb3ab570c28b8a1f2dd80a15f.pdf.

[11] J. Chen, Y. Tong, W. Gray, and Q. Ji, "A robust 3D eye gaze tracking system using noise reduction", in *Proceedings of the 2008 symposium on Eye tracking research & applications - ETRA '08*, New York, New York, USA: ACM Press, 2008, p. 189, ISBN: 9781595939821. DOI: 10.1145/1344471.1344518. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1344471.1344518.

[12] Kang Wang and Q. Ji, "Real time eye gaze tracking with Kinect", in *2016 23rd International Conference on Pattern Recognition (ICPR)*, IEEE, Dec. 2016, pp. 2752–2757, ISBN: 978-1-5090-4847-2. DOI: 10.1109/ICPR.2016.7900052. [Online]. Available: http://ieeexplore.ieee.org/document/7900052/.

[13] E. D. Guestrin and M. Eizenman, "Remote point-of-gaze estimation requiring a single-point calibration for applications with infants", in *Proceedings of the 2008 symposium on Eye tracking research & applications - ETRA '08*, New York, New York, USA: ACM Press, 2008, p. 267, ISBN: 9781595939821. DOI: 10.1145/1344471.1344531. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1344471.1344531.

[14] Y. G. Shin, K. A. Choi, S. T. Kim, C. H. Yoo, and S. J. Ko, "A novel 2-D mapping-based remote eye gaze tracking method using two IR light sources", in *2015 IEEE International Conference on Consumer Electronics, ICCE 2015*, IEEE, Jan. 2015, pp. 190–191, ISBN: 9781479975426. DOI: 10.1109/ICCE.2015.7066375. [Online]. Available: http://ieeexplore.ieee.org/document/7066375/.

[15] S. W. Shih and J. Liu, "A Novel Approach to 3-D Gaze Tracking Using Stereo Cameras", *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 34, no. 1, pp. 234–245, Feb. 2004, ISSN: 10834419. DOI: 10.1109/TSMCB.2003.811128. [Online]. Available: http://ieeexplore.ieee.org/document/1262497/.

[16] T. Ohno, N. Mukawa, and A. Yoshikawa, "FreeGaze", in *Proceedings of the symposium on Eye tracking research & applications - ETRA '02*, New York, New York, USA: ACM Press, 2004, p. 125, ISBN: 1581134673. DOI: 10.1145/507072.507098. [Online]. Available: http://portal.acm.org/citation.cfm?doid=507072.507098.

[17] J. G. Wang and E. Sung, "Study on eye gaze estimation", *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 32, no. 3, pp. 332–350, Jun. 2002, ISSN: 10834419. DOI: 10.1109/TSMCB.2002.999809. [Online]. Available: http://ieeexplore.ieee.org/document/999809/.

[18] H. Yamazoe, A. Utsumi, T. Yonezawa, and S. Abe, "Remote gaze estimation with a single camera based on facial-feature tracking without special calibration actions", in *Proceedings of the 2008 symposium on Eye tracking research & applications - ETRA '08*, New York, New York, USA: ACM Press, 2008, p. 245, ISBN: 9781595939821. DOI: 10.1145/1344471.1344527. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1344471.1344527.

[19] C. Morimoto, A. Amir, and M. Flickner, "Detecting eye position and gaze from a single camera and 2 light sources", in *Object recognition supported by user interaction for service robots*, vol. 4, IEEE Comput. Soc, 2003, pp. 314–317, ISBN: 0-7695-1695-X. DOI: 10.1109/icpr.2002.1047459. [Online]. Available: http://ieeexplore.ieee.org/document/1047459/.

[20] S. S. Mohapatra and K. Kinage, "Iris tracking using a single web-cam without IR illumination", in *Proceedings - 1st International Conference on Computing, Communication, Control and Automation, ICCUBEA 2015*, IEEE, Feb. 2015, pp. 706–711, ISBN: 9781479968923. DOI: 10.1109/ICCUBEA.2015.144. [Online]. Available: http://ieeexplore.ieee.org/document/7155939/.

[21] Y. G. Shin, K. A. Choi, S. T. Kim, and S. J. Ko, "A novel single IR light based gaze estimation method using virtual glints", *IEEE Transactions on Consumer Electronics*, vol. 61, no. 2, pp. 254–260, May 2015, ISSN: 00983063. DOI: 10.1109/TCE.2015.7150601. [Online]. Available: http://ieeexplore.ieee.org/document/7150601/.

[22] T. F. Cootes, G. J. Edwards, and C. J. Taylor, "Active appearance models", in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1407, Jun. 1998, pp. 484–498, ISBN: 3540646132. DOI: 10.1007/BFb0054760. [Online]. Available: http://ieeexplore.ieee.org/document/927467/.

[23] G. Edwards, A. Lanitis, C. Taylor, and T. Cootes, "Statistical models of face images — improving specificity", *Image and Vision Computing*, vol. 16, no. 3, pp. 203–211, Mar. 1998, ISSN: 02628856. DOI: 10.1016/s0262-8856(97)00069-3. [Online]. Available: https://www-sciencedirect-com.focus.lib.kth.se/science/article/pii/S0262885697000693.

[24] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees", in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2014, pp. 1867–1874, ISBN: 9781479951178. DOI: 10.1109/CVPR.2014.241. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6909637.

[25] X. Cao, Y. Wei, F. Wen, and J. Sun, "Face alignment by explicit shape regression", *International Journal of Computer Vision*, vol. 107, no. 2, pp. 177–190, 2014, ISSN: 15731405. DOI: 10.1007/s11263-013-0667-3. [Online]. Available: http://www.jiansun.org/papers/CVPR12%7B%5C_%7DFaceAlignRegression.pdf.

[26] P. Dollar, P. Welinder, and P. Perona, "Cascaded pose regression", 2010, pp. 1078–1085. DOI: 10.1109/cvpr.2010.5540094. [Online]. Available: https://pdollar.github.io/files/papers/DollarCVPR10pose.pdf.

[27] T. Hastie, R. Tibshirani, and J. Friedman, "Linear Methods for Regression", in, 2009, pp. 43–99. DOI: 10.1007/978-0-387-84858-7_3. [Online]. Available: http://link.springer.com/10.1007/978-0-387-84858-7%7B%5C_%7D3.

[28] Z. R. Cherif, A. Naït-Ali, J. F. Motsch, and M. O. Krebs, "An adaptive calibration of an infrared light device used for gaze tracking", *Conference Record - IEEE Instrumentation and Measurement Technology Conference*, vol. 2, pp. 1029–1033, 2002. DOI: 10.1109/IMTC.2002.1007096. [Online]. Available: http://ieeexplore.ieee.org/document/1007096/.

[29] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection", in *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, vol. I, IEEE, 2005, pp. 886–893, ISBN: 0769523722. DOI: 10.1109/CVPR.2005.177. [Online]. Available: http://ieeexplore.ieee.org/document/1467360/.

[30] C. Burges and A. Smola, *Advances in kernel methods - support vector learning, http://svmlight . joachims.org*, 1999. [Online]. Available: https://dl.acm.org/citation.cfm?id=299094%20http://svmlight.joachims.org.

[31] Kishor Datta Gupta, *Automatic pupil detection and extraction by c# | Kishordgupta's Blog*, 2010. [Online]. Available: https://kishordgupta.wordpress.com/2010/12/18/automatic-pupil-detection-by-c/ (visited on 05/27/2019).

[32] W. admin, *Understanding the Differences Between C#, C++, and C - C# Station*, 2018. [Online]. Available: https://csharp-station.com/understanding-the-differences-between-c-c-and-c/ (visited on 06/05/2019).

[33] Visualstudio, *Visual Studio IDE, Code Editor, VSTS, & App Center - Visual Studio*, 2019. [Online]. Available: https://visualstudio.microsoft.com/ (visited on 06/06/2019).

[34] AForge, *AForge.NET :: Computer Vision, Artificial Intelligence, Robotics*, 2013. [Online]. Available: http://aforgenet.com/ (visited on 05/23/2019).

[35] D. King, *dlib C++ Library - Introduction*, 2009. [Online]. Available: http://dlib.net/ (visited on 05/23/2019).

[36] Math.net Christoph Rüegg, *Math.NET Numerics*. [Online]. Available: https://numerics.mathdotnet.com/ (visited on 05/23/2019).

[37] D. F. DeMenthon and L. S. Davis, "Model-based object pose in 25 lines of code", in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 588 LNCS, Kluwer Academic Publishers, Jun. 1992, pp. 335–343, ISBN: 9783540554264. DOI: 10.1007/3-540-55426-2_38. [Online]. Available: http://link.springer.com/10.1007/BF01450852.

Appendix A

Manipulated data

In this appendix, the diagrams that show the offset of the estimations in relation to the actual points on the screen are presented. The center of each diagram is where the screen point would be and the estimated points are displayed at their horizontal and vertical distance from the screen point. Out of the 810 data points recorded in total, 149 (around 18.4%) are outside the plots displayed below, as they deviate far too much for the plotting to be intuitive if they were to be included.

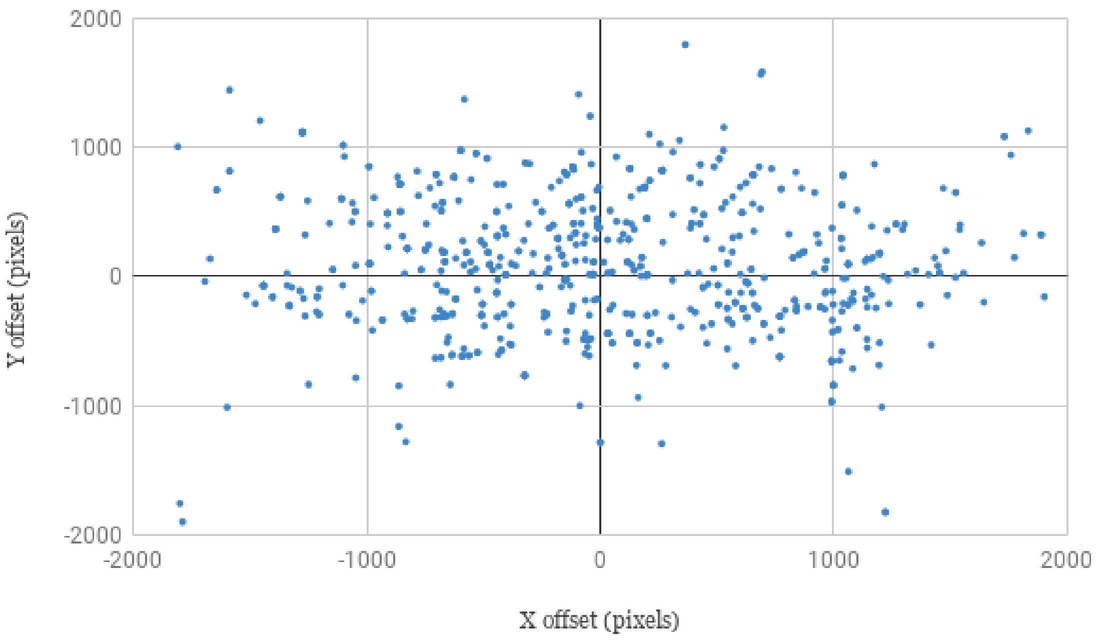


Figure A.1: All subjects' offset data.

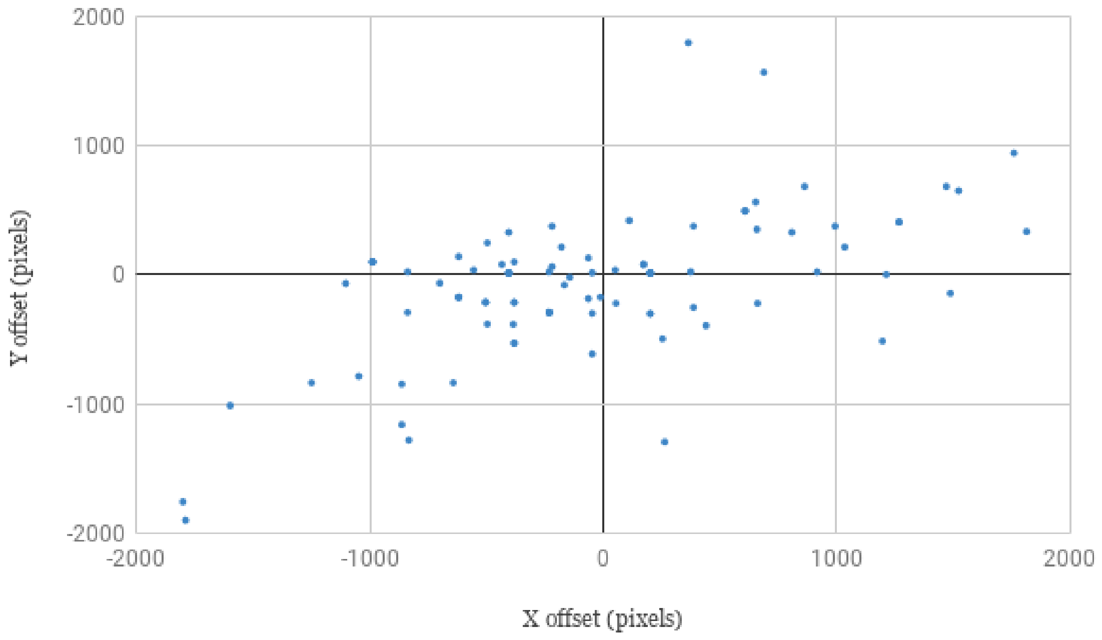


Figure A.2: Subject #1's offset data.

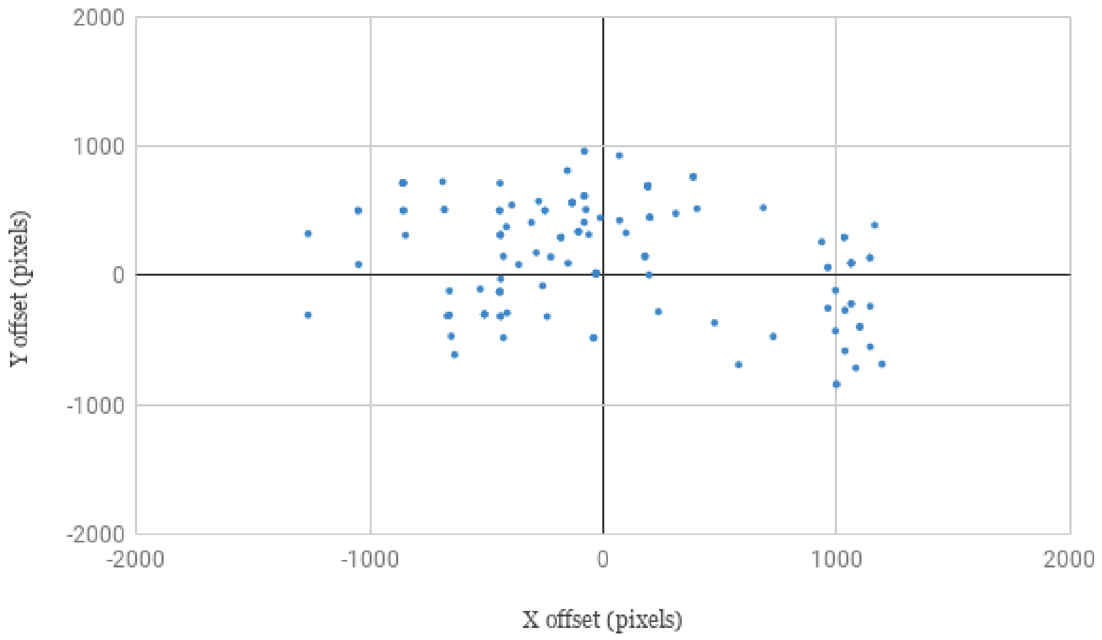


Figure A.3: Subject #2's offset data.

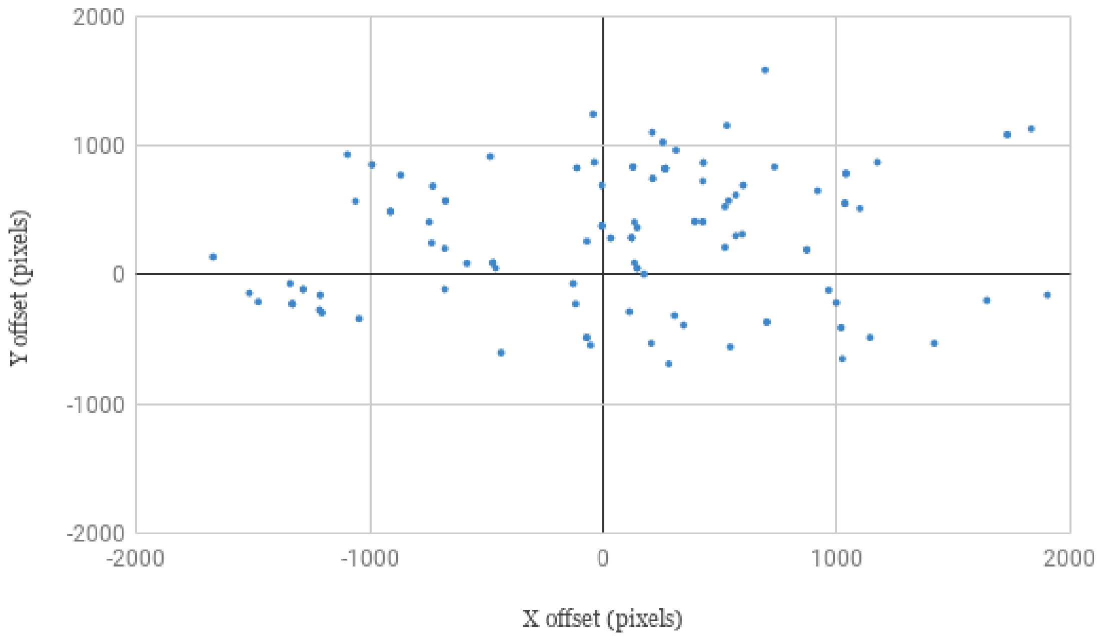


Figure A.4: Subject #3's offset data.

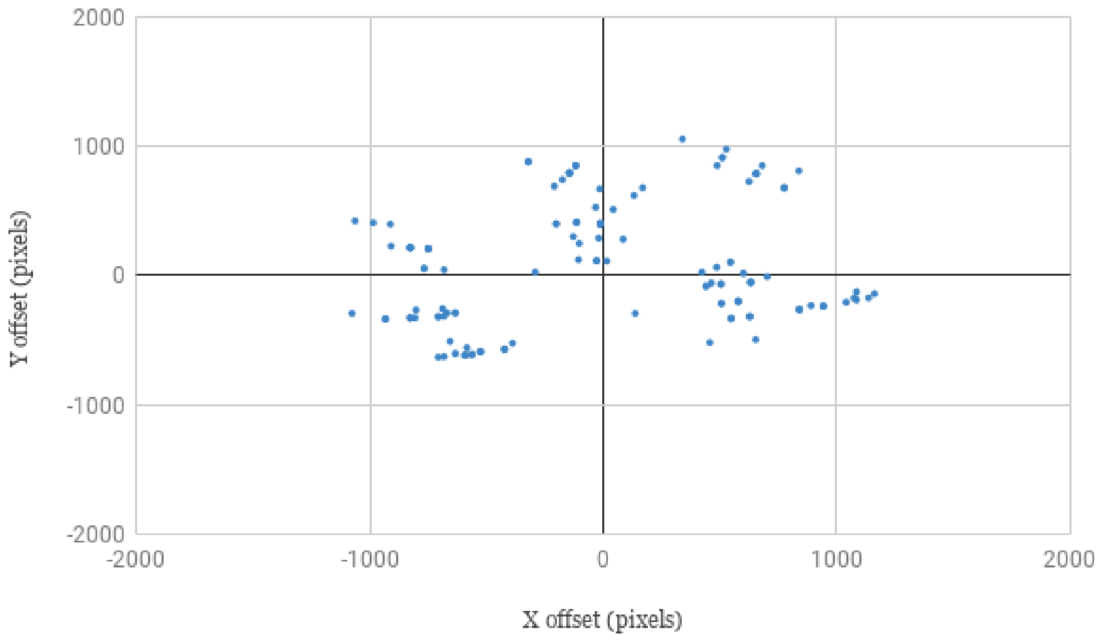


Figure A.5: Subject #4's offset data.

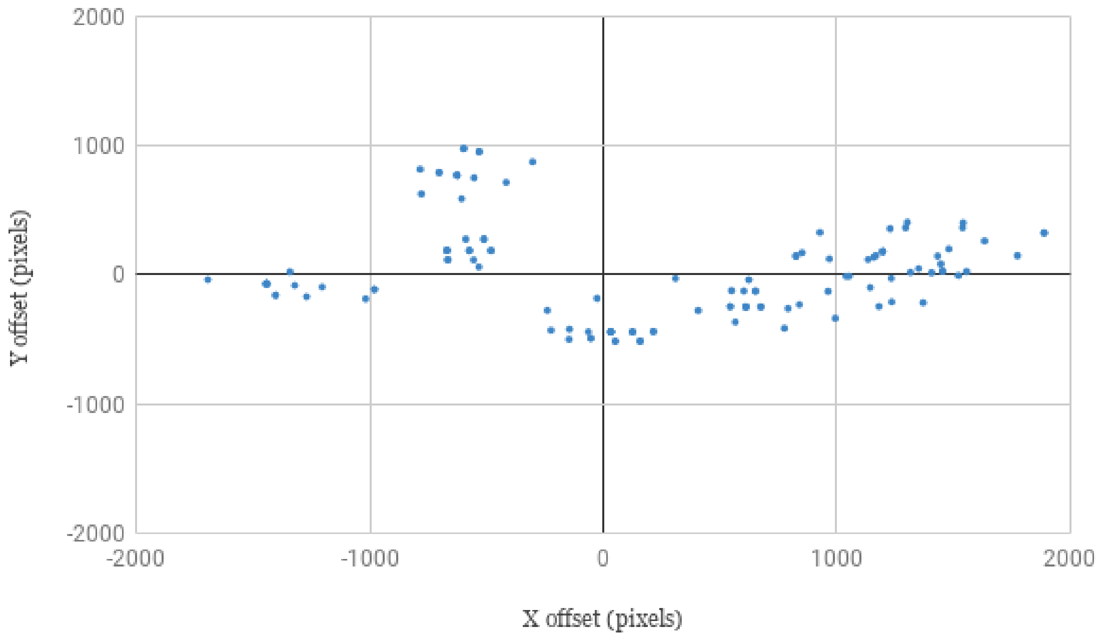


Figure A.6: Subject #5's offset data.

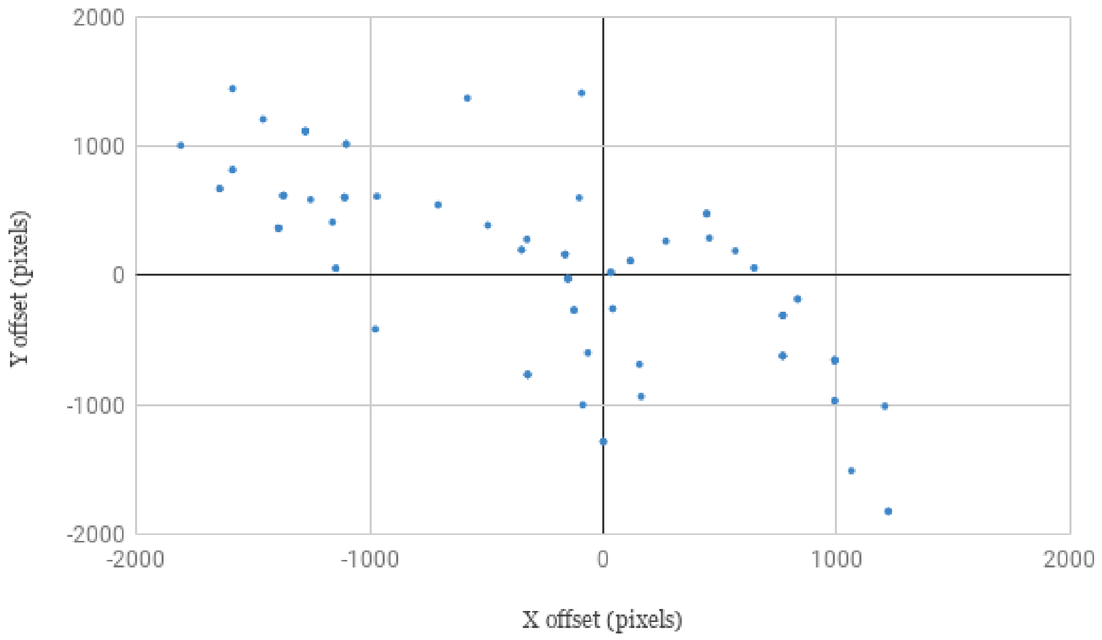


Figure A.7: Subject #6's offset data.

TRITA TRITA-EECS-EX-2019:168