Exploring Eye Tracking Techniques On Smartphones

Iosif Karkanis
Abstract

Exploring Eye Tracking Techniques On Smartphones

Iosif Karkanis

Eye tracking is a major field in medical sector, especially in psychiatry for giving an insight of the patients with mental disorders. Nowadays, the existence of mobile devices with powerful hardware can provide an opportunity to investigate if it is possible to track the eye using these devices without using any additional hardware.

This thesis will try to explore the possibility of tracking the center of the pupil using mobile devices. To achieve that, the eye tracking algorithms of template matching and eye detection using image gradients, are implemented in these devices. The application was also implemented as a background service and as a stand-alone activity in order to investigate the performance and usability for these two methods.

Both template matching and eye detection using image gradients algorithms show promising results in terms of performance and accuracy respectively.

Handledare: Edith Ngai
Ämnesgranskare: Anders Hast
Examinator: Justin Pearson
IT 15052
Tryckt av: Reprocentralen ITC
I would like to express my deepest appreciation to my supervisor Edith Ngai and my reviewer Anders Hast for their continuous guidance and persistent help during this thesis until the very end of it.

Also, I would like to thank Fotis Papadopoulos from Department of Neuroscience, Psychiatry, University Hospital of Uppsala for his cooperation.

Last but not least, many thanks go to my friends for their useful feedback and help, during my thesis.

Finally, I would like to thank my family for their support and trust during my whole studying career. Without them, I would not have reached this point.
Contents

Acknowledgements

Contents

List of Figures

Abbreviations

1 Introduction

1.1 Motivation

1.2 Goal Of The Thesis

1.3 Thesis Outline

2 Background and Related Work

2.1 Background

2.2 Related Work

2.3 Industrial Products

3 Tools

3.1 Android

3.1.1 System Architecture

3.1.2 Activity Lifecycle

3.1.3 Android Service

3.2 OpenCV

3.3 Hardware

4 Tracking Algorithms

4.1 Haar Feature-based Cascade Classification

4.2 Template Matching Algorithm

4.3 Timm’s and Barth’s Algorithm Using Image Gradients

5 System Implementation

5.1 OpenCV in Activity

5.2 OpenCV as a Service

5.3 Template Matching Implementation

5.4 Timm’s and Barth’s Algorithm Implementation

5.5 Calibration
6 Evaluation
6.1 Template Matching Test ................................. 28
6.2 Timm’s and Barth’s Algorithm Test ..................... 30
6.3 Calibration Test ........................................ 34

7 Discussion ........................................... 37

8 Conclusions ........................................... 39
8.1 Conclusion ............................................. 39
8.2 Future Work ......................................... 40

Bibliography ........................................... 41
List of Figures

2.1 Tobii Eye Tracking Products ........................................ 6
2.2 Eye Tribe .............................................................. 6
2.3 Interactive-minds Eye Tracking Products ....................... 6
2.4 Smart Eye, Eye Tracking Products ................................. 7

3.1 Android System Architecture ........................................... 9
3.2 Android Activity Lifecycle ............................................. 10
3.3 Android Service Lifecycle ............................................. 11
3.4 Android OpenCV Manager ............................................. 13
3.5 OpenCV Manager Install ............................................. 13

4.1 Haar Features .......................................................... 16
4.2 Template matching method ............................................ 17
4.3 Slide the template for matching ..................................... 17

5.1 Example of the calibration scaling method ....................... 27

6.1 Example of template matching algorithm detecting correctly the eyes . . . 29
6.2 Example of template matching algorithm detecting the eye with low accuracy ................................................. 29
6.3 Example of template matching algorithm with no accuracy at all at one of the eyes ................................................. 30
6.4 An example of Timm’s and Barth’s algorithm pupil detection ................................. 31
6.5 Example of Timm’s and Barth’s algorithm while looking to the left ........... 31
6.6 Example of Timm’s and Barth’s algorithm while looking to the right ........... 32
6.7 Example of Timm’s and Barth’s algorithm without reflection on the pupil 32
6.8 Example of Timm’s and Barth’s algorithm having a reflection on the pupil 33
6.9 Timm’s and Barth’s algorithm detection from 1m distance ..................... 33
6.10 Example of Timm’s and Barth’s algorithm in a dark environment ............. 34
6.11 Calibration tests on a 5 inch smartphone .................................. 35
6.12 Calibration tests from 5 inch smartphone scaled to screen’s resolution .... 35
6.13 Calibration tests on a 10 inch tablet ........................................ 36
6.14 Calibration tests on a 15 inch monitor ........................................ 36
6.15 Calibration tests on a 24 inch monitor ........................................ 36
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALS</td>
<td>Amyotrophic Lateral Sclerosis</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>FPS</td>
<td>Frames Per Second</td>
</tr>
<tr>
<td>NDK</td>
<td>Native Development Kit</td>
</tr>
<tr>
<td>OpenCV</td>
<td>Open Computer Vision</td>
</tr>
<tr>
<td>OpenGL</td>
<td>Open Graphics Library</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

One of the most important sensors of the body is the eye. Tracking the eye in real-time is important to understand how people interact and perceive the world. Eye tracking is useful at marketing, gaming industry, and personal health. The focus in health is into Parkinson’s disease, psychiatric disorders, head injuries and others[1]. Eye movement examination can provide unique insight on a person’s mental and neurological state by providing information about where that person looks at and by analyzing eye’s movement. Eye tracking can be useful to study social interaction[2] in i.e autism, anxiety disorders[3] and even diagnosis of schizophrenia[4].

Existing eye tracking solutions rely on expensive equipment, which are only available in specialized laboratories, thus the daily and repeated measurements of eye movements are not feasible at home or in the daily environment. With the advancement of mobile sensing technology, eye-tracking applications on mobile devices have great potential to be used as an advanced medical device for assisting the diagnosis and follow-up of mental disorders. For example, such applications could be used in the daily clinical work as an assisting tool in the diagnostic evaluation of patients. Another potential could be a directed screening of population for such symptoms of mental disorders in order to facilitate an early detection and diagnosis, which often warrant a better prognosis. Finally, the monitoring of patient for alterations of such eye-movements patterns alongside with treatment may be useful clinical tool for the follow-up of patients.
This thesis was proposed for from my supervisor Edith Ngai a senior lecturer at Division of Computer Systems Department of Information Technology, Uppsala University in cooperation with Fotis Papadopoulos which is a senior lecturer at Department of Neuroscience, Psychiatry, University Hospital. The interesting part of this project is to investigate how technology and especially mobile devices can contribute the medicine on the research of mental disorders. Since eye tracking is a major field for psychiatry, having a device that will be used by the patients without having to visit a doctor, was the main idea. The idea of combining technology with medicine, motivated me to investigate this question.

1.2 Goal Of The Thesis

The purpose of this study is to develop an application which will track in real-time the eye’s movement by making the user to interact with the application. The first step is to investigate how accurate and stable are the results of the eye tracking on mobile devices. Tests will be done using simple algorithm based on pictures taken by the cameras on mobile devices of different sizes. The application is going to be implemented for Android devices.

1.3 Thesis Outline

The thesis contains some background around eye tracking area in Chapter 2. The background shows some research that took place in academia and companies that work on eye tracking technologies. The tools used for the implementation of the thesis are presented in Chapter 3. Chapter 4 explains the algorithms of template matching and detection using image gradients, which were used for detecting the center of the pupil and in Chapter 5 the implementation of these algorithms for the mobile application is analyzed. The results of the implementation tests are shown in Chapter 6. Finally, in Chapter 7 reside a discussion of the results, possible improvements and future work.
Chapter 2

Background and Related Work

2.1 Background

Eye tracking is a science that has been studied for over a century. The means for examinations were, direct observation in the beginning and using optical tools. Nowadays, at the digital era, tracking is held by using sophisticated systems. The first time that it was used, showing important differences in printing and screens, was at early 90’s. USA TODAY[5], using Gallup Applied Science’s eye tracking system, filmed the eye of a regular fan of National Football League (NFL) and that of a football analyst. Then a computer tracked where the eye was focusing on the TV. A cursor was implanted after that to the film to show where the eye was focused.

Most of the eye trackers today are based on infrared cameras. This kind of equipment use infrared/near-infrared light to calculate the corneal reflections[6] and thus detect the pupil of the eye.

There are three main methods of eye movements: [7].

- Saccades
- Smooth Pursuit
- Fixation
Saccades are fast movements of the eye, resulting in a reposition of the latter in the environment. This movement can be caused by reflection or voluntarily changing the focus of the sight.

Smooth pursuit is the movement of the eye when it focuses on a moving object. The eye adjusts the velocity of the movement depending on the velocity of the object that is focused on.

Fixation is the method by which the eye focuses on a fixed point. In fixation, some slight eye movements like tremor, drift and microsaccades cannot be avoided due to neurological reasons.

Smooth pursuit and fixation are the most popular methods when it comes to medical research. The model that is going to be used for this thesis, is the model of fixation.

2.2 Related Work

Eye tracking and generally tracking other features of the human face is an area that got the attention of many researchers for many years and is still under development. Some of the research that has been done on this field is going to be explained in this section.

According to Cristinacce et al.[8] there is a multi-stage algorithm that can track face features automatically. In this approach the first stage is to detect the face. The second stage is to apply and combine individual feature detectors using an algorithm known as Reinforcement of Feature Responses. The last stage is to refine the points that were predicted, using Active Appearance Model.

Another method that was used from Asteriadis et al.[9] is to locate the eyes on a face, geometrically. In this method a face detector is applied, in order to find the bounding box of the face, and as result the edge map is extracted. Subsequently, a vector is assigned to every pixel, pointing to the closest edge pixel. These vectors and specifically their slope and length are used for tracking the eye.

Turkan et al.[10] is introducing eye localization using edge projection. In this algorithm a high-pass filter of a wavelet transform is being used to filter the face. As a result, the edges of the face are highlighted and a caricature-like representation is obtained. In the
new filtered image, horizontal projections and profiles of the edge region are analyzed in order to provide the candidate points for each eye. All these candidate points are classified using a support vector machine based classifier. The most probable candidate points will provide the eyes’ locations.

Most of the eye tracking solutions use infrared cameras on their hardware, which makes the tracking precise. Valenti and Gevers [11] though tried to find a solution without infrared cameras, just by using low resolution web cameras. The algorithm is using isophotes to gain invariance to linear lighting changes, such as contrast and brightness, in order to achieve rotational invariance and low computational cost.

Another method of tracking the eye, used for low resolution images like web cameras, is the one introduced by Timm and Barth [12]. The algorithm is using the gradients of the image to track the eye. A function is derived using only dot products. The maximum of this function is the point that the gradient vectors intersect and thus the center of the eye. One of the first methods using image gradients, was implemented by Kothari and Mitchell [13]. In comparison with Kothari’s method, Timm’s algorithm is taking care of the problems that arise due to eyebrows, eyelids and glasses.

2.3 Industrial Products

Many of the companies in the industry are working on eye tracking systems, producing their own hardware and software for this purpose. These companies target at many areas like gaming, transportation, marketing and health care. So far some of the companies are:

- Tobii[14]
- The Eye Tribe[15]
- Interactive Minds[16]
- Smart Eye[17]

Tobii is probably the most well-known company specialized in eye tracking, based in Stockholm, Sweden. They design their own hardware and software for the products.
They also include speech recognition to some of them. The focus is on a variety of fields such as gaming, health care, transportation, user experience, market research, performance assessment on sports, augmentative alternative communication. The products can be portable like glasses[2.1a] or a small piece of hardware[2.1b], connected to a device, or embedded to a device[2.1c].

Another company that focuses in eye tracking is Eye Tribe, a danish company from Copenhagen, manufacturing an affordable portable device for computers, tablets and smartphones[2.2].

Interactive-minds is one of the companies that is specialized in eye tracking, mostly for medical reasons like ALS patients and people with Locked-in syndrome. The products are portable that can be adjusted to the monitors. These are monocular[2.3a], binocular[2.3b] and with four cameras[2.3c].
Smart Eye is a Swedish company regarded as a global leader in head movement and eye tracking for Automotive, Aviation and Aerospace research. The products are portable[2.4a] or embedded to a monitor[2.4b]. Two of the products are measuring the eyelid closure, head pose and gaze tracking and are used for automotive, in the driver’s cabin[2.4d, 2.4c].

These companies produce software and hardware for eye tracking. The hardware, usually, includes a high-resolution camera with a large field-of-view, used to capture images of the user, required for eye tracking. It also includes, Near Infra-Red Light-Emitting Diodes (NIR-LEDs) that are used to generate even lighting and reflection patterns in the eyes of the user.
Chapter 3

Tools

3.1 Android

The project will be implemented for mobile devices. The major operating systems for mobile devices are the following three:

- Android OS
- iOS
- Windows Phone OS

Android OS [18] was chosen among the other two mobile operating systems because of the free tools that it offers in order to develop the application (eclipse, Android Studio). Development of android can take place in any operating system the developer desires (Mac OS, Windows, Linux). For iOS application and Windows Phone application development, Mac OS and Windows OS is needed respectively.

Furthermore, Android is an open source OS, without the limitations that iOS and Windows phone OS have due to closed source code. Moreover, it is not so easy to publish the application to the market in Windows and iOS platforms because of the companies policies.

For these reasons Android OS seems to be the optimal choice for the purpose of this project.
3.1.1 System Architecture

Android system’s architecture consists of four main layers as shown in figure[3.1]. At the bottom layer is the Linux kernel which provides the basic functionality including memory management, device management like camera, display, etc. as well as the device drivers. The next layer is the Libraries layer, which consists of several libraries responsible for audio and video recording and playing, OpenGL framework for the graphics etc. In the same layer lies the Android Runtime, where the core libraries and Dalvik Virtual Machine reside. The latter is a kind of Java Virtual Machine designed specifically for Android OS. The third layer is the Application Framework which provides services to an application in the form of Java classes. The last layer which is Applications, is the one in which applications can be installed.

![Android System Architecture Diagram](image)

**Figure 3.1:** Android System Architecture

Android OS is written mainly Java while the drivers are written in C language. When it comes to development, Java is the primary programming language that every application is written but there is the possibility to use C/C++ due to the Native Development Kit (NDK).
### 3.1.2 Activity Lifecycle

Activity is an application component lying in the Application Framework. This component provides a screen for the user to interact with. An application can have multiple activities in it. Every activity has a lifecycle that can be seen in figure 3.2.

![Android Activity Lifecycle Diagram](image)

**Figure 3.2: Android Activity Lifecycle**

In order for an activity to exist, it has to be created first by calling the `onCreate()` function. Then `onStart()` function is called in order for the activity to start running. Also `onResume()` is called to continue the application running that can be paused with `onPause()`. If the user or the system kills the application (for example, the user presses
the back button) the application will pause and then resume if it will be opened again. By pressing the back button or killing manually the application, `onStop()` and `onDestroy()` are called.

### 3.1.3 Android Service

A Service is another application component. This component does not need the interaction of the user. It is a long-running operation, running on the background. The component runs even if the user switches to another application. The lifecycle of a service is shown in the figure[3.3].

![Android Service Lifecycle Diagram](image-url)

**Figure 3.3:** Android Service Lifecycle
A service can have two forms. The first form is that it can be started from an application and run indefinitely while the second form is to be bounded to an other application component. Even though figure[3.3] shows two different lifecycles, these two forms can be combined. The service can be started using `onStartCommand()` or `onBind()` functions. The difference is that by using `onStartCommand()`, the application still runs until it is destroyed, and `onBind()` lifecycle ends by calling `onUnbind()`.

### 3.2 OpenCV

Open Source Computer Vision Library (OpenCV)[19] is an open source library, providing an infrastructure for computer vision based applications. OpenCV also accelerates the application according to the machine’s architecture. The interfaces provided by OpenCV are in C++, C, Python, Java and Matlab, supporting Windows, Linux, Android, MacOS and iOS operating systems.

OpenCV has developed an application that can be found on the Google Play Store, called OpenCV Manager. OpenCV Manager is an android service, targeted to manage OpenCV library binaries on the device that is using an application, developed using OpenCV. The manager provides hardware optimizations for all the supported platforms and less memory usage, achieved by having all the libraries on one application. In that way, every OpenCV application does not have to store the libraries individually, but calling the libraries needed as a service. Figure[3.4] shows the usage model for the end user using OpenCV Manager.
The application will try to bind to OpenCV libraries and ask for installation of the manager if it is not already installed as depicted in figure 3.5. After the installation it will find and load the necessary libraries and initiate OpenCV. Then the application is ready to run.

Figure 3.4: Android OpenCV Manager

Figure 3.5: OpenCV Manager Install
3.3 Hardware

The hardware used for this project are mobile Android-based devices.

- Samsung Galaxy S3
- Samsung Galaxy S4
- Samsung Galaxy Tab 2

<table>
<thead>
<tr>
<th></th>
<th>Galaxy S3</th>
<th>Galaxy S4</th>
<th>Galaxy Note 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Android 4.3</td>
<td>Android 5.0.1</td>
<td>Android 4.2.2</td>
</tr>
<tr>
<td>CPU</td>
<td>Quad-core 1.4GHz</td>
<td>Quad-core 1.6GHz</td>
<td>Dual-core 1GHz</td>
</tr>
<tr>
<td></td>
<td>Cortex-A9</td>
<td>Cortex-A15</td>
<td>Cortex-A9</td>
</tr>
<tr>
<td>Memory</td>
<td>1GB RAM</td>
<td>2GB RAM</td>
<td>1GB RAM</td>
</tr>
<tr>
<td>Display</td>
<td>4.8 inches</td>
<td>5.0 inches</td>
<td>10.1 inches</td>
</tr>
<tr>
<td>Resolution</td>
<td>720x1280</td>
<td>1080x1920</td>
<td>800x1280</td>
</tr>
<tr>
<td>Camera</td>
<td>1.9MP, 720p, 30fps</td>
<td>2MP, 1080p, 30fps</td>
<td>VGA, 640p</td>
</tr>
</tbody>
</table>
Chapter 4

Tracking Algorithms

In this chapter, the algorithms that were implemented for tracking the head and then the center of the pupil, are going to be explained in theory to have an understanding of the way they work.

4.1 Haar Feature-based Cascade Classification

Detection of an object using cascade classifiers was proposed from Viola and Jones [20] and Lienhart and Maydt [21] improved it a bit later. A machine learning approach was used to train a classifier from many positive and negative images. Then the classifier is used to detect objects in other images. In order for the classifier to find an object that the dimensions are smaller than the original image, every location of the image has to be check using the classifier.

An advantage of the classifier is that it can scale when it needs to find an object instead of resizing the original image. When an object that needs to be found has unknown size, the scan procedure should be done several times at different scales.

The cascade classifier consists of many simpler classifiers that are applied subsequently to the targeted region of interest until one of them is rejected or all of them are passed. Haar-like features are shown in figure [4.1]. The features which a classifier uses, are specified by the shape, the position and the scale.
1. Edge Features

(a)  (b)

2. Line Features

(a)  (b)

3. Four-rectangle Features

(a)

**Figure 4.1: Haar Features**

### 4.2 Template Matching Algorithm

In computer vision, the term template matching is a technique for finding areas of an image that are similar to a template image.

In this technique there are two primary components:

- Source image
- Template image

Source image \((I)\) is the original image in which the template is applied, in order to find a specific area. Template image \((T)\) is the image that is used for the comparison with the source image in order to detect the best matching area as shown in figure[4.2].

In order to identify the matching area, the template image has to move on top of the source image, by one pixel at a time as shown in figure[4.3]. On every pixel several
calculations take place, to show if the point from the template image is a good match to the respective point of the source image at that specific time. To do that the metric value of the comparison is stored in a matrix (R).

There are several methods for template matching in OpenCV. The one that was used for this thesis is correlation coefficient [22].

$I(x,y)$ is a value of source image pixel and $T(x,y)$ a value of template pixel at location $(x,y)$. The width of the template image in pixels is denoted as $w$ and the height of the template image as $h$.

$T'$ stands for the average value of pixels in the template and $I'$ the average value of pixels in the current window of the source image, where $x' = 0 \ldots w - 1$ and $y' = 0 \ldots h - 1$. 
The result matrix $R$ is calculated by summing up the product $T'(x', y')$ and $I'(x + x', y + y')$ for each overlay position shown in equation 4.1.

$$R(x, y) = \sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))$$  \hspace{1cm} (4.1)

where:

$$T'(x, y) = T(x', y') - \frac{1}{(w \cdot h) \cdot \sum_{x'', y''} T(x'', y'')}$$  \hspace{1cm} (4.2)

$$I'(x + x', y + y') = I(x + x', y + y') - \frac{1}{(w \cdot h) \cdot \sum_{x'', y''} I(x + x'', y + y'')}$$  \hspace{1cm} (4.3)

### 4.3 Timm’s and Barth’s Algorithm Using Image Gradients

A method used for detecting the center of the pupil is the one suggested by Timm and Barth [12].

In order to find the center of a circular object, which in that case is the pupil, image gradients vectors are used. A possible center depends on the orientation of these vectors. Assuming $c$ is a possible center and $g_i$ (equation 4.4) is the gradient vector at position $x_i$, then $d_i$ (equation 4.5) which is the normalized vector needs to have the same orientation with the gradient vector $g_i$.

$$g_i = \begin{pmatrix} \frac{\partial I(x_i, y_i)}{\partial x_i} \\ \frac{\partial I(x_i, y_i)}{\partial y_i} \end{pmatrix}^T$$  \hspace{1cm} (4.4)

$$d_i = \frac{x_i - c}{\|x_i - c\|_2}, \hspace{1cm} \forall i: \|g_i\|_2 = 1$$  \hspace{1cm} (4.5)

where:

$$\frac{\partial I(x_i, y_i)}{\partial x_i} = \frac{I(x + 1, y) - I(x - 1, y)}{2}$$  \hspace{1cm} (4.6)
\[
\frac{\partial I(x_i, y_i)}{\partial y_i} = \frac{I(x, y + 1) - I(x, y - 1)}{2}
\]  

(4.7)

The possible center \( c^* \) is calculated by finding the dot products between the normalized vector \( d_i \) and the gradient vector \( g_i \) at the pixel \( x_i \), \( i \in \{1, \ldots, N\} \) (equation 4.8).

\[
c^* = \arg\max_c \left\{ \frac{1}{N} \sum_{i=1}^{N} (d_i^T g_i)^2 \right\}
\]  

(4.8)

In order to obtain equal weight for every pixel position, both the \( d_i \) vectors and the corresponding \( g_i \) should be scaled to unit length to improve lighting and contrast.

The possible center estimation might be wrong sometimes due to eye glasses and eyelids. In order to correct that, a weight \( w_c \) is applied so the dark possible centers are chosen instead of the bright ones. \( w_c \) is the gray value of the inverted input image \( I^* \), as shown at (equation 4.9)

\[
w_c = I^*(c_x, c_y)
\]  

(4.9)

Also a Gaussian filter is applied to avoid reflections of glasses and generally bright spots.

The possible center is calculated by using the weight as shown at (equation 4.10).

\[
c^* = \arg\max_c \left\{ \frac{1}{N} \sum_{i=1}^{N} w_c(d_i^T g_i)^2 \right\}
\]  

(4.10)

This is all the necessary information needed when the picture obtains the image of the eye. Sometimes hair, glasses and eyebrows show image gradients with different direction in comparison with these from the pupil. So a post processing step is going to help with that. A threshold is applied in order to remove all the gradient values from the borders of the image. The maximum of these values is used to determine the center of the pupil.
Chapter 5

System Implementation

The implementation of this thesis project took place on Android OS devices. The programming languages that were used are Java and C++ using OpenCV libraries as mentioned in Chapter 3.

OpenCV for Android has implemented two methods for connecting to the camera. One of the methods is by using Android’s SDK camera call and the other is the one implemented by OpenCV in C++ for working with almost any hardware and not only in android.

By using the Java implementation the advantage is that every android phone can work with OpenCV, without depending on the hardware or the Android version.

The implementation that uses the native method, meaning native for OpenCV, written in C++ has the drawback that does not work in every smartphone and needs a lot of time to update the OpenCV in order for the new versions of Android to run. For example the native version is not working with Android Lollipop. The advantage of C++ in comparison with java is that C++ is known for the performance that offers, even though needs more attention than Java while developing.

5.1 OpenCV in Activity

OpenCV for Android can be used in an Activity to show the results directly to the screen. In order to load the OpenCV libraries, OpenCV Manager is going to do that,
asynchronous initialization is implemented. After the initialization finishes, `OnManager-Connected` callback will be called in UI thread. OpenCV calls can not be used before this callback. `BaseLoaderCallback` treats the application context as Activity and in case of the initialization fails, it calls `Activity.finish()`.

```java
public class EyeTrackingActivity extends Activity implements CvCameraViewListener {
    private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
            switch (status) {
                case LoaderCallbackInterface.SUCCESS:
                    Log.i(TAG, "OpenCV loaded successfully");
                    mOpenCvCameraView.enableView();
                    break;
                default:
                    super.onManagerConnected(status);
                    break;
            }
        }
    }

    @Override
    public void onResume() {
        super.onResume();
        OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_9, this, mLoaderCallback);
    }
    ...
}
```

Listing 5.1: Asynchronous Initialization

To call the surface on the UI to show the picture, the XML file for the UI needs to have this:
Also permission has to be granted so the application can use the camera. To do that, the code below has to be added in the `AndroidManifest.xml`:

```xml
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera" android:required="false"/>
<uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
<uses-feature android:name="android.hardware.camera.front" android:required="false"/>
<uses-feature android:name="android.hardware.camera.front.autofocus" android:required="false"/>
```

Listing 5.3: Grant access to camera

The following functions are used for the implementation:

```java
public void onCameraViewStarted(int width, int height) {
}

public void onCameraViewStopped() {
}

public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
    return inputFrame rgba();
}
```

Listing 5.4: Main methods for OpenCV in Android

`onCameraViewStarted` is used to make action when the camera is initialized. Usually matrices are initialized in this method.
onCameraViewStopped is used when the camera is stopped and usually the matrices are released when this method is called.

onCameraFrame is capturing a frame from the camera and return it to the screen. This is were all the calculations are taking place.

5.2 OpenCV as a Service

OpenCV can run as a service too, but this is more tricky than running in an activity. Using the existing methods is not possible since they are implemented in such a way, that they will run on an activity.

The native method is the more suitable choice according to OpenCV documentation because it creates the camera, destroys it and delivers the frames while camera is still enabled.

To do that, a modification in the NativeCameraView.java from OpenCV is needed. The main methods needed are:

```java
protected boolean connectCamera(int width, int height) {
}

private boolean initializeCamera(int width, int height) {
}

protected void disconnectCamera () {
}

private void releaseCamera () {
}
```

Listing 5.5: Main methods for OpenCV as a service

Then the class which is responsible for retrieving the frames of the camera, is implemented.

```java
private static class NativeCameraFrame {
```
private VideoCapture mCapture;
private Mat mRgba;
private Mat mGray;

public NativeCameraFrame(VideoCapture capture) {
    mCapture = capture;
    mGray = new Mat();
    mRgba = new Mat();
}

public Mat rgba() {
    mCapture.retrieve(mRgba, Highgui.CV_CAP_ANDROID_COLOR_FRAME_RGBA);
    return mRgba;
}

public Mat gray() {
    mCapture.retrieve(mGray, Highgui.CV_CAP_ANDROID_GREY_FRAME);
    return mGray;
}

public void release() {
    if (mGray != null) mGray.release();
    if (mRgba != null) mRgba.release();
}

};

Listing 5.6: Class to retrieve frames from camera

Finally, the last class that the service needs is a runnable thread, in order for the frame to be delivered continuously. In this class the calculations are going to be added in order to take place for every frame.

private class CameraWorker implements Runnable {
    NativeCameraFrame nativeCamera = new NativeCameraFrame(mCamera);
    public void run() {
        do {
            if (!mCamera.grab()) {
                break;
            }
        } while (!mStopThread);
    
}
5.3 Template Matching Implementation

The method of template matching that was explained in chapter 4, was implemented in Java using OpenCV Java API.

The first step is to detect the face using Haar cascades as mentioned in chapter 4. The cascades is a machine learning based approach where cascade function is trained from a lot of positive and negative images. The classifier extracts features which in this case is the face.

In OpenCV the function to do that is `detectMultiScale()` which detects the face using the classifier `haarcascade_frontalface_alt.xml`. The size of the area that needs to be searched can also be imported in the function.

Then according to the dimensions and coordinates of the face detected, two squares are calculated to represent the area of the eyes.

The last step is to get the template of the eyes in the first five frames from the time the face is detected, and use this template for finding the eyes. The functions responsible for this are:

- `get_template()`
- `match_eye()`

Function `get_template()` has as an input a classifier which will detect the eyes using `haarcascade_lefteye_2plits.xml` using the function `detectMultiScale()` mentioned before, the area of the eye and the size of the template image.

Finding the eye area the algorithm is searching for the darkest point which is the iris and creates a rectangle that will be the template. This is how the template is created.
Function `match_eye()` has as input the area of the eye and the template that created at the first five frames. `Imgproc.matchTemplate` is the OpenCV function to match the template to the original image using the method explained in chapter 4.

All of the above functions are called in `onCameraFrame()` function so that the result will be returned to the screen after every frame.

### 5.4 Timm’s and Barth’s Algorithm Implementation

Timm’s and Barth’s algorithm in finding the center of the pupil is implemented using Java for Android development and C++ for the calculations. In comparison with the template matching algorithm that was implemented purely in Java, this one is implemented using C++ because of the amount of calculations needed for this algorithm. OpenCV is written in C++ and has a lot of functions that are not available in the Java API yet. Also C++ is a language that provides acceleration and better performance. So C++ was chosen for this implementation.

To begin with, the Java part is almost similar with the one followed in template matching. It loads the `haarcascade_frontalface_alt.xml` and detects the face using `detectMultiScale()` function. After that there is a function to construct the rectangle area for the eyes. From this function the C++ part is called. The calculations for detecting the center of the pupil take place in the C++ part.

The gradients are being computed using a function called `computeMatXGradient()`. The gradients in this function are calculated as shown to the equations 4.6 and 4.7 for x and y coordinates respectively explained in chapter 4.

Then a threshold is applied as mentioned at chapter 4 to follow the normalization of the gradients (equation 4.5).

In order for the calculation of the weight $w_c$, a Gaussian blur is applied and the image is being inverted using `GaussianBlur()` from OpenCV.

Then the equation for finding the center of the eye is applied to find the possible centers for each gradient location. From these values for possible centers, OpenCV function `minMaxLoc()` will find the maximum point which will be the center of the pupil.
5.5 Calibration

The idea of calibration is to create a rectangle, given the coordinates of the center of the pupil at the time the user is looking to every corner of the screen. In that way the range of the center of the pupil is measured in that rectangle. The rectangle will be at the end a scaled image of the device’s dimensions as shown in figure 5.1.

![Figure 5.1: Example of the calibration scaling method](image)

In order to do that the center of the eye is stored in an array by pressing the button that exist at every corner of the screen. When someone is touching something is looking at the finger at that moment. By pressing the button, the user is automatically looking at the corner of the screen where the buttons are placed.

The last step is to scale the rectangle to the screen size.
Chapter 6

Evaluation

Tests were made on Android mobile devices as mentioned in chapter 3. The performance of the algorithms depends on the processing power of the devices as also on the resolution of the camera.

6.1 Template Matching Test

Template matching seems to be a fast operation with not many computations. The normal frames per second rate for the camera working without any computations is almost 16. The frame rate drops around 7-10 FPS due to the computations, but the accuracy of the algorithm is good enough. Figures 6.1, 6.2 and 6.3 show that the algorithm can sometimes detect sufficiently the eye, even though most of the times the focus is lost without detecting the pupil. The green box indicates the detected face area, the red box is the eye region of interest and the yellow squares shows the result of the template matching which is searching for the pupil.
Figure 6.1: Example of template matching algorithm detecting correctly the eyes

Figure 6.2: Example of template matching algorithm detecting the eye with low accuracy
Figure 6.3: Example of template matching algorithm with no accuracy at all at one of the eyes

6.2 Timm’s and Barth’s Algorithm Test

Image gradients is a slow operation due to the amount of calculations. On a colored image with high resolution, FPS is dropping significantly to less than one. Some optimizations were done to increase the performance. Changing to gray scale was one of these changes because the operations occur on gray scale image, so we removed the transformation from colored to gray scale. Screen resolution is a major factor on the performance, the lower the better. Also in the beginning the cascades files were loaded from C++ and when that moved to Java it was faster.

Even, after the optimizations and also since the program was written mostly in C++, were supposed to make it fast, the results were not satisfying. The algorithm was running at maximum 2 FPS at the highest image resolution under normal lightning conditions.

The accuracy on the other hand is much more better than the one from template matching, as it detects the eye with high accuracy. The result is shown in figure [6.4]. In every picture the big box represents the face region detected using haar-cascades classifier, the smaller boxes show the region of every eye and a small circle is the result of detecting the center of the pupil.
Chapter 6. Evaluation

Figure 6.4: An example of Timm’s and Barth’s algorithm pupil detection

Even when the eye is looking in different points the accuracy is still acceptable with the FPS being really low as shown in figures [6.5] and [6.6]

Figure 6.5: Example of Timm’s and Barth’s algorithm while looking to the left

The accuracy is better when there is no reflection on the pupil but even with reflection it is still close enough to the actual center as shown in figures [6.7] and [6.8].

Even when the face is further from the camera, the precision is good.[6.9]
Figure 6.6: Example of Timm’s and Barth’s algorithm while looking to the right.

Figure 6.7: Example of Timm’s and Barth’s algorithm without reflection on the pupil.
Figure 6.8: Example of Timm’s and Barth’s algorithm having a reflection on the pupil

Figure 6.9: Timm’s and Barth’s algorithm detection from 1m distance
When the environment is dark it can still detect the center of the pupil almost good enough. These deviations are justified when not even the human eye can detect the center fairly as shown in figure [6.10].

![Example of Timm’s and Barth’s algorithm in a dark environment](image)

**Figure 6.10:** Example of Timm’s and Barth’s algorithm in a dark environment

### 6.3 Calibration Test

In order to have an acceptable calibration scale, the eye was re-sized in the screens dimensions, so that it will have more wide area for the eye to move. Even though that seemed to work fine while watching the camera, the calibration was not producing the expected results as shown in figure [6.11]. The expected shape was rectangle but only once the calibration gave something close enough to a rectangle.

The test was also held while the head was standing in order to stay still. The same happened with the device so the image would be steady. Even under these conditions the eye was moving around as also the head.

To show the difference from the calibration test, in figure 6.12, one can see the size of the previous calibration tests in contrast with the resolution of the screen.
Some experiments were made for bigger screens like a 10 inch tablet, a 15 inch monitor and a 24 inch monitor, as shown in figures 6.13, 6.14 and 6.15 respectively.
Figure 6.13: Calibration tests on a 10 inches tablet

Figure 6.14: Calibration tests on a 15 inch monitor

Figure 6.15: Calibration tests on a 24 inch monitor
Chapter 7

Discussion

During the project two of the available algorithms introduced in chapter 2, were implemented and tested. There are many other algorithms to be tested that might provide both accuracy and great performance on mobile devices due to the amount of the computations needed.

The main idea of calibration was to take the position of the center of the eye in pixels and create a scaled rectangle of the screen according to the user’s stare. The problems in this was that the screen is small (5.0 inches) which implies that the distance of the pixels of the center looking to the two sides of the device is really close. Also, Timm’s and Barth’s algorithm detect the eye with high accuracy, one can see that the algorithm in every pixel is detecting the center of the pupil in different pixels than before and that is because the algorithm might find suitable as center one of the other possible centers explained in the algorithm. Another problem was the movement of the head. Even though, both the device and the head were stable, when we zoomed in to the face area we saw that the head was trembling and that was changing the pixels position. So the head was not so steady. When the face detection was on or tried to keep the head in a steady position without face recognition, the face was moving again a bit, so the does not matter if the face detection is on or not. All of the above lead us to the conclusion that the calibration was not feasible in order to have something working as we saw on Chapter 6. This deviations on the pixels had as a result some of the corners of the expected rectangle to collapse or even have different orientation. The results were irregular shapes and maybe triangles. The same tests were made for bigger screens up
to 24 inches. The results show that the bigger screen the easier to distinguish the shapes but still for the reasons we mentioned above the results are not always sufficient.

Regarding the code, some optimizations need to be done in order to get better performance. In the case of running the detection algorithm as a service, trying to open the camera and capture the frames for the calculations to happen, using a method different from the one proposed from OpenCV, for example using Android’s SDK methods, could be a good way to try it so it will work on every phone. Running as a service could also be a plus for performance because the image does not need to be shown in the screen, which is power consuming. Also, not using the screen will save some of the processing power.

Because of the time limit of this project not every method could be implemented and tested. Another method for testing, that might show better results will be to implement a mathematical model for mathematical construction of the eyeball explained from Jian-Gang Wang et al. [23]. The result of this method would be, a vector showing where the eye is looking at. To do that we need the center of the eye and some other key points of the eye. So, detection of the center of the pupil is still an important characteristic. Also another method would be the construction of a 3D model of the face by capturing pictures of the last from multiple angles. Then in every frame the eye could mathematically calculate the 3D image of the eyes which seems similar to the infrared technology for eye tracking.

There are many things to be improved, especially for mobile devices where the screen size is small. The accuracy for this devices is important and difficult without using infrared cameras.
Chapter 8

Conclusions

8.1 Conclusion

The purpose of the thesis was the implementation of an Android application which tracks the eye of the user. The first step to achieve that, is to detect the region of the face which was possible using haar-cascades classifiers. To track the center of the eye two rectangles were positioned relative to the face detection rectangle to create the region of every eye. In the eye regions two methods were applied, the one is template matching and the second is the algorithm proposed from Timm and Barth. Both of the methods were also implemented as an activity as also as a background service. Running the detection algorithms as a service has the drawback of not being able to run on every Android device but might gain on performance in comparison with the activity. Template matching results were not to accurate as Timm’s and Barth’s algorithm but the performance was better than the last one. Timm’s and Barth’s algorithm was used for the tests, because of the high accuracy it provided.

The calibrations tests were failing because of the small area the eye was moving, in order to look at the corners of the device. The algorithm was moving by few pixels and since the head was trembling, something that was visible after zooming in the eye, irregular shapes were generated instead of a rectangle.

The conclusion of all the thesis is that this method of calibration is failing on small devices at least. The possibility the calibration to succeed in large screens is bigger,
because of the wider range of the movement of the eye. Other methods would be more appropriate for mobile devices but need to be tested first.

8.2 Future Work

The next step would be for smartphones to try a different method of calibration like the one proposed from Jian-Gang Wang et al. [23]. In his method the construction of the 3D model of the eye is required mathematically. Finding some key points and assuming that the center of the eye ball is steady, the gaze is detected.

Also another method would be to take pictures of the face from different angles and construct a 3D model of the head. Then a mathematical model could be applied, so that the background could be extracted and the calculations will take place only in the area of the face instead of the whole image. Also calculating the curves from the 3D model, would help detecting the eye better. This will probably provide with more accuracy and better performance.
Bibliography


