Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Usage of Eye Tracking for Adaptive Contextual Applications

Jakub Gruber

Supervisor: Ing. Jiří Šebek
May 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Gruber** Jméno: **Jakub** Osobní číslo: **456916**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Využití sledování očí pro adaptivní kontextově závislé aplikace

Název bakalářské práce anglicky:

Usage of eye-tracking for adaptive contextual applications

Pokyny pro vypracování:

Sledování očí je technika, při které jsou měřeny oční pohyby jednotlivce tak, aby vědce znal jak tam, kde se člověk dívá v daný čas, tak pořadí, v němž se oči člověka přesouvají z jednoho místa do druhého. Sledování pohybů očí lidí může pomáhat vědcům HCI pochopit vizuální a zobrazení založené na zpracování informací a faktory, které mohou mít vliv na použitelnost systémových rozhraní. [2]
Cíle práce jsou následující:
1) Prostudujte možnosti využití sledování očí pro použití v mobilních aplikacích [1,2]
2) Vytvořte framework, který bude poskytovat rozhraní pro využití dat ze sledování očí
3) Vytvořte mobilní aplikaci demonstrující funkčnost frameworku
4) Otestujte aplikaci na uživatelích
5) Upravte aplikaci na základě potřeb uživatelů
6) Vyhodnoťte testování, možné problémy a přínosy.

Seznam doporučené literatury:

1. D. Selvathi, N. Dhivya, ""Realization of VLSI architecture to detect driver drowsiness for road accident avoidance system"", Green Engineering and Technologies (IC-GET) 2016 Online International Conference on, pp. 1-5, 2016.
2. Ghaoui, Claude. Encyclopedia of human computer interaction. Hershey PA: Idea Group Reference, 2006. ISBN 9781591405627.
3. Šebek, J. - Richta, K.: Usage of Aspect-Oriented programming in Adaptive Application Structure. New Trends in Databases and Information Systems: ADBIS 2016 Short Papers and Workshops, BigDap, DCSA, DC, Prague, Czech Republic, August 28-31, 2016, Proceedings

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek, Software Engineering and Networking FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **16.01.2018** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgements

I would like to thank my supervisor Ing. Jiří Šebek for providing materials as well as for his support, patience and advice during writing the thesis.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 25, 2018

. . . . . .

Abstract

The aim of this bachelor thesis is to design and implement a framework for mobile applications that will be collecting passive information about the user using a technique of eye tracking.

“Eye tracking is a technique whereby an individual’s eye movements are measured so that the researcher knows both where a person is looking at any given time and the sequence in which the person’s eyes are shifting from one location to another. Tracking people’s eye movements can help HCI researchers to understand visual and display-based information processing and the factors that may impact the usability of system interfaces.” [1]

Another goal will be to implement a mobile application that will be based on the developed framework adapt user interface.[2]

Keywords: eye tracking, adaptive user interface, dynamic authentication, Android, context

Supervisor: Ing. Jiří Šebek

Abstrakt

Cílem bakalářské práce je navrhnout a implementovat framework pro mobilní aplikace, který bude shromažďovat pasivní informace o uživateli pomocí sledování očí.

"Sledování očí je technika, při které jsou sledovány oči jedince tak, že výzkumníci vědí, kam se jedinec v daný okamžik dívá a zároveň také znají sekvenci, při které jedincovy oči přechází z jednoho místa na druhé. Sledování očních pohybů může HCI výzkumníkům pomoci pochopit vizuální informace a informace na obrazovce a faktory, které mohou ovlivnit použitelnost systémových rozhraní." [1]

Dalším cílem bude implementovat mobilní aplikaci, která bude na základě vyvinutého frameworku adaptovat uživatelské rozhraní.[2]

Klíčová slova: sledování očí, adaptivní uživatelské rozhraní, dynamická autentizace, Android, context

Překlad názvu: Využití sledování očí v kontextuálně adaptivních aplikacích

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

Nowadays, almost everyone owns a smartphone and carries it every day in a backpack or a pocket, on a way to school, to work or when is on a vacation. Wherever a user is, he has a smartphone with him. This provides developer a lot of possibilities to collect a variety of information that can be later user to create a personalized application.

Mobile devices are used in a various environment, in various conditions and most importantly by various users. Every used is unique and has different needs.

As a result, an application should, in the best case, be able to adjust itself based on a combination of situation, environment and user preferences. A personalized and adaptive application should also predict future actions and try to save some time for a user by suggesting something that is expected to be required in future or by changing its UI to make content easily accessible.

Context changes very often and therefore such an action must be detected, predicted and executed immediately. Because of that, it is necessary to have a source of information that will be fast and reliable enough.

Sight is one of the most used and the most important human senses and it is also a great source of information for situations mentioned above. By watching user's eye movement, a developer can get necessary information immediately and automatically amend application behavior or UI.

Considering these advantages, one would expect eye tracking to be used on a daily basis in a mobile application, the contrary is the case.

1.2 Goals

The goals of this bachelor thesis are to investigate what current solutions are available for developers who would like to take advantage of eye tracking in their applications, evaluate existing solutions, find their problems and set metrics to improve.

Based on previous analysis, design and implement framework that will eliminate problems of existing solutions and provide an approach that will

simplify eye tracking integration.

The framework should be designed with an emphasis on an adaptive user interface.

1.3 Structure

The first part of this thesis explains basic terms that are required to understand the topic.

The second part mentions frameworks, applications and technologies where eye tracking is used.

The next section describes analysis and design of framework and reasons why particular solutions were chosen instead of others.

Implementation details of a framework are described in a section that follows. It contains an overview how to integrate framework into an existing application, what classes and modules are available to developer and recommendations to preserve performance.

Following part mentions possible use cases of Eye Tracker Framework in real applications. Use cases that are implemented in sample application contain also screenshots.

Final chapters explain how to install the framework, what approach was chosen for testing and whether it resulted in expected results. It also summarizes possible advantages, disadvantages, overall success and the future of the thesis.

Chapter 2
Background

2.1 Adaptive User Interface

AUI is defined as *a software artifact that improves its ability to interact with a user by constructing a user model based on partial experience with that user*. [3] It means that it is changing its components automatically without any input needed from the user, only by previous experience and behavior. However, AUI still needs inputs that are later evaluated and are reasons for the change. These inputs are provided by the underlying framework.

Such as input is usually a change to contextual information, which is an information that is related to a specific context. As a specific context can be understood the process of tracking user's eye movements, his presence in front of a tracker or simply a time at a given moment, location or period of a year.

Context is responsible for assigning different meaning to collected data. Meaning of data can be changed with a new context but does not necessarily has to.

As mentioned above, data need to be linked with contextual information. Classic UI can be considered as a holder for information, but as the context changes, information should too. Common UI has no tools to detect these changes. By using adaptive UI, the displayed information is changed on the go as the context changes.

Support for an adaptive UI is not in common development tools and therefore it is necessary to use an underlying framework. A problem with these frameworks is that they are not primarily designed for this purpose.

It results in a difficult integration, mixing of business logic with a logic related to framework handling and code base is flooded with unnecessary framework-related calls. As the time goes, source code becomes unmaintainable.

2.2 Human-Computer Interaction

Human-Computer Interaction is a discipline that studies communication between human and computer and evaluates design and realization. [4]

It utilizes the whole communication and its goal is to create UI which is pleasant and straightforward to use.

Based on best practices, every application's UI should be developed in order to meet following factors [7] and be:

- useful - only relevant content is presented
- usable - it is easy to use
- desirable - every element of the UI has its own purpose
- findable - content in the UI is easy to locate, it is where users expect that to be
- accessible - content is reachable for people with disabilities
- credible - content is trustworthy

2.2.1 Types of HCI interfaces

HCI interfaces could be for purpose of this thesis divided into two groups similarly to a common division of brain-computer interfaces [1].

Active

In a traditional active BCI, a user tries to actively send commands to an application. [19] Application then reacts to commands sent by the user and adjust its UI or executes the desired action.

An advantage of this approach is that user knows exactly what he is trying to achieve and what should be expected result.

An example eye tracking UI that could be considered as active are all applications that allow a user to fully control a device by eye movements. These applications simulate computer mouse in Android devices and replace a default Android keyboards.

User synchronizes keyboard in the beginning and later uses only eye movements to navigate within an application, enter text and click buttons.

A category of active UI is used in applications that are designed to also support handicapped people, which is something that tends to be forgotten nowadays.

Passive

In comparison to active BCI, passive ones try to analyze user's brain activity to predict future actions and execute them without an input required [19].

Goals of passive BCI and AUI are very similar as both are focused on automatic adaptation without direct inputs and therefore this thesis and developed framework will focus on supporting passive AUI.

[1] An interface that connects a device to a brain to make the device controllable by the brain.

2.3 Eye Tracking

Eye tracking is a technique that is widely used to understand the cognitive process of an individual performing assigned task. [5].

Within this process, observers concentrate mainly on the length of the eye gaze at a given point, paths that examined subject chooses while searching for another point of interest or the length of attention that subject is able to pay.

2.3.1 Usages

Eye tracking has a huge potential and therefore it is already used in multiple areas, mainly in cognitive science, design, automotive and it starts to be used in HCI.

Human-Computer Interaction

Eye tracking is used in HCI testing to analyze desktop or web applications. A user is placed in front of a computer with an opened application and is browsing through.

Researchers observe the whole process and record scanpaths [2] created by analyzed subject. Scanpaths are later converted into heatmaps [3].

Heat maps are adjusted according to selected metrics mentioned below:

- Where are users looking in order to find important elements
- How long are they looking to analyze how much attention they can expect from users
- In which order are users looking at elements
- How placement or size can affect attention
- What parts distract users
- If different user groups browse application in a different order

Design

Results from HCI research mentioned above are used as inputs for graphic designers who need to adjust application's UI to make it more readable and easy to use.

[2] A graph that visualizes the way how the user was changing point of interests.

[3] A graph that uses a color scale to visualize mostly visited areas on the website/application.

Automotive

Driver drowsiness and distraction are responsible for most of the car accidents nowadays. [6]

Because of that, automotive companies integrate eye trackers in their cars. An eye tracker is placed facing a driver and consists of a camera and a special software that analyses captured video in real time.

While driving, eye tracker is able to analyze gaze patterns and changes in blink duration and based on that detect whether a driver is drowsy or not. In that case, car systems recommend a driver to take a break.

2.3.2 Ways of Tracking

Devices that are designed to track user's eye movements are called eye trackers.

Eye trackers are either specialized devices that were made with single purpose to track user's eyes or they can be a simple phone that has a front facing camera and integrated SW to process a video.

Every eye tracker works on a slightly different principle.

Head-Mounted HW

Head-mounted HW technique requires an external device to be mounted on user's head. It contains a sensor that communicates with a connected device that is able to convert data from a sensor into eye movement information.

This solution is very accurate and can also easily track eye movement paths and create scanpaths.

Common eye trackers look like a single camera attached to user's head, but newer ones are designed to look more like glasses. Example of a newer version of head-mounted eye tracker can be seen in Figure 2.1

Unfortunately, it is not suitable for common use cases because head-mounted HW is very expensive and is not comfortable to wear the whole day. The user will also know that he is tracked and therefore can behave differently than he would in normal cases. This change of behavior has a negative impact on results of tracking and it is important to remember that while evaluating results.

It is mainly used for laboratory testing of desktop or web applications. Applications are later redesigned according to results of testing to show important content on places that user is looking in the first place. Using this technique, developers can create efficient UI that holds only necessary information.

Near-infrared Camera

Near-infrared cameras are used in the most precise eye trackers that are available on the market. These trackers are placed on a table under a screen and are facing user's face or they are integrated into glasses that can later track user's eye movements in a similar way like head mounted HW.

Figure 2.1: HW eye tracker (Undertook from [8])

Tracking works in the way that near-infrared light is directed towards eye pupil and creates reflections in a cornea. These reflections are tracked by a camera and evaluated by underlying HW. Because of the need of near-infrared camera, this technique is not applicable to Android devices.

Simplified anatomy of an eye is displayed in Figure 2.2

Video Processing

Video processing is focused on low resources and fast processing. It uses a real-time video processing to process single video frames. Frame processor applies cascade classifiers [4] to each frame and tries to detect features. These features contain information about a position of user's face, ears, eyes, mouth and other facial landmarks such as nose or ears.

Video processing has many advantages. Probably as the biggest advantage could be considered the fact that it does not require any external HW except camera to provide frames to frame processor. The technique is also reasonably accurate for basic purposes therefore it can easily be used in mobile applications that cannot waste resources and need to stay low on the price. The accuracy of results may also be increased by the fact that user does not need to know that he is tracked, therefore he will not change his behavior.

On the other side, accuracy is negatively influenced by many factors that lower final results. The biggest problem is a light that causes reflections and

[4] Components to detect features in a frame that are iteratively trained by positive and negative samples.

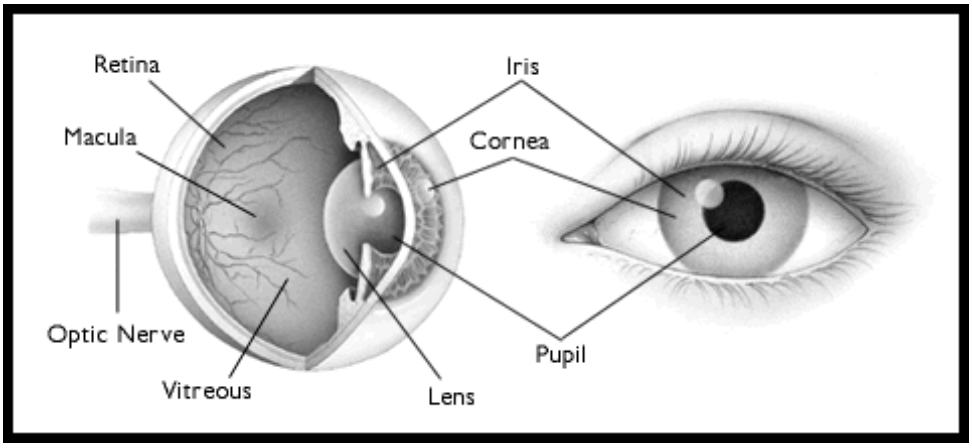

Figure 2.2: Eye Structure depicting cornea and pupil location (Undertook from [12])

shadows on the face and makes it more complicated for cascade classifiers to locate desired features of a face.

2.4 Android

Modern mobile devices run on Android OS, iOS, Blackberry OS, Windows Mobile or some minor OS. As can be seen in Figure 2.3 Android is the most used operating system nowadays and according to the expectations, it still will be in the near future.

Even though it is very fragmented as can be seen in Figure 2.4, it's SDK is written in Java and provides backward compatibility for most of the functionalities. Because of that, frameworks written for Android have a wide variety of potential developers that could make use of a developed framework.

2.4.1 Activity

A basic building block of every Android application is called Activity. Activity represent a single screen of application that implements one real-world activity that user can do, e.g. sing-in, buy tickets, create a new note, take a picture.

Every standalone application must have one activity that is marked as a so-called *main activity*. This activity is created as a first one as is an entry point to the whole application.

An activity must not be instantiated by developer directly, but it is done by the OS. OS creates instances of activities, handles background processes and user inputs. To be able to do that, activity has to provide an implementation of lifecycle methods. The most common ones are listed below:

- *onCreate*
  - Called when a system wants to create new activity instance. The best place where to initialize UI components. Must be always

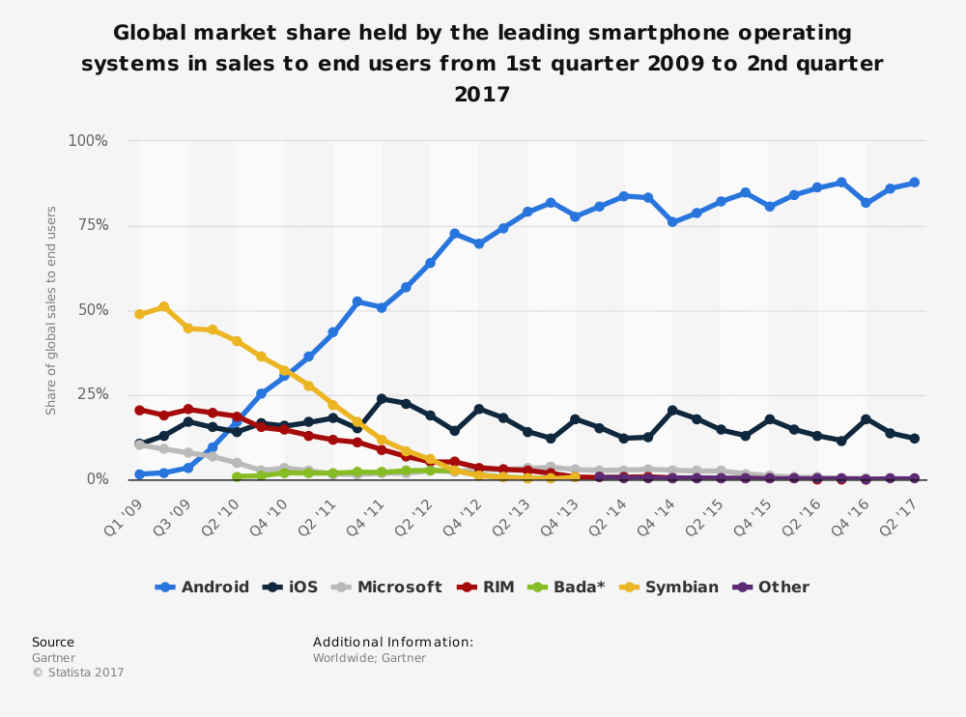

Figure 2.3: Mobile OS market share between 2009-2017. (Undertook from [9])

implemented.

- *onResume*
  - Called right after *onStart* method (not listed here). The method is also called after *onPause*.
- *onPause*
  - Called when activity loses focus. An activity must not necessarily be destroyed when this method was called, but it can follow. *OnPause* is invoked if user taps e.g. 'Back' button.
- *onDestroy*
  - Called before system destroys activity. A suitable place to release resources that are available to the single application at given time, e.g. camera.

More detailed diagram depicting activity lifecycle methods and its hierarchy can be seen in Figure 2.5.

2.4.2 Fragment

In comparison with activity, fragment serves as a reusable part of a screen that is attached to parent activity. A fragment cannot exist on its own in the system and also cannot serve as an entry point to an application.

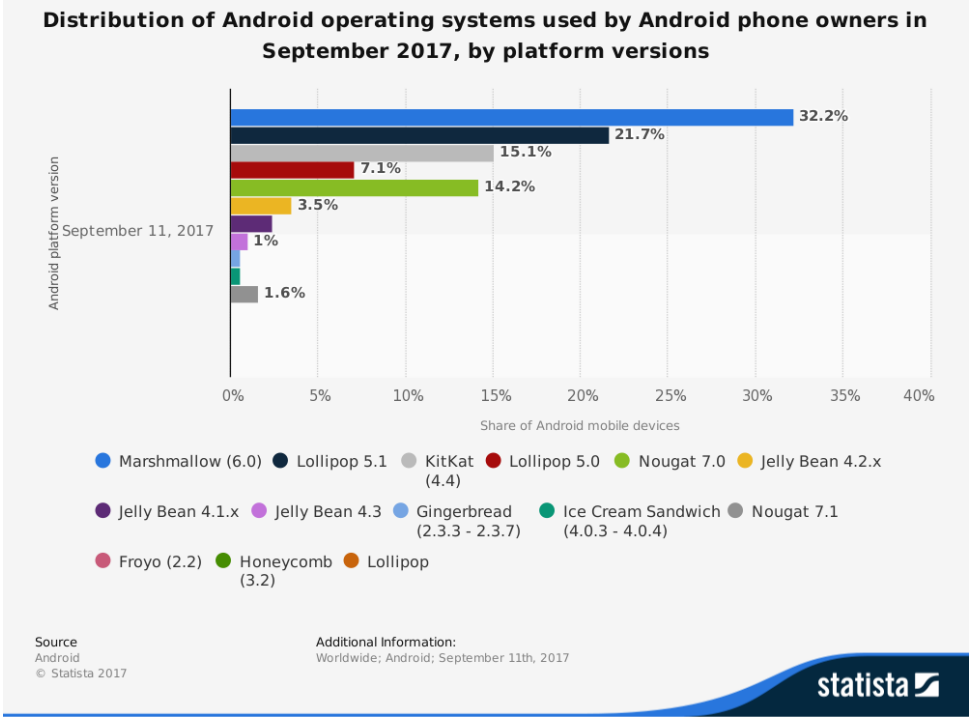

Figure 2.4: Android OS version market share in 2017. (Undertook from [10])

A fragment has similar lifecycle to activity, but it adds a few methods such as *onAttach*, *onCreateView*, *onActivityCreated* or *onDetach*. These methods are meant to be used in order to synchronize its internal state with parent activity which fragment is attached to.

As already said, it can and even must run as a part of an activity. It's lifecycle is synchronized and cannot run on its own.

However, a fragment can be running hidden in the background of activity in so-called 'headless mode'.

Headless mode means that fragment does not create any UI which results in invisible component running in the background.

In comparison with simple classes, its main advantage is that it can access system resources such as a camera. It can also easily respond to events triggered by the user, e.g. closing application, pressing back button etc.

2.5 Existing Solutions

There are no existing solutions that would fulfill requirements for eye tracking framework with a focus on adaptive UI. Only possible ways how to achieve that is to use computer vision framework that enables video processing and create a wrapper that provides expected functionality.

Doing so is quite complicated and usually scares off potential developers that might want to use it and want to start quickly.

Next section describes two major eye tracking frameworks and explains its

advantages or disadvantages.

2.5.1 OpenCV

An open source computer vision library that is highly optimized and meant to be used for computational difficult image processing. It works on a low level and offers a basic set of functions that need to be merged together to create complex applications. OpenCV is usually main building block for commercial eye tracking SW.

On the other side, the library is written in C++ which requires complicated configuration to get it available in Android and also expects C++ knowledge from developers. Calls to the library are made either using provided Java wrappers from OpenCV or by writing own C++ source codes that are later called from Java using a *native* keyword.

Integration of OpenCV is complicated. A developer has to configure the project in order to load and import shared-object library when building an application, import Java wrappers, explicitly open the camera and then start parsing each camera frame using OpenCV's cascade classifiers. On the other side, it provides more flexibility.

Since Android API 21, Android SDK has a new way how to work with camera and because of that, the developer has to support both old and new API in order to support older devices.

Previously mentioned steps are making integration unnecessary complicated.

2.5.2 Google Mobile Vision

Mobile Vision is a library developed by Google that is focused on computer vision. It provides functions and structures for barcode or text scanning and also face recognition. The library is distributed using Gradle repository and later, when is installed to target device, it downloads additional data from Google Play Services.

Integration into existing application or library project is not as difficult as using OpenCV, but it still has its downsides.

A developer needs to take care of instances that belong to Mobile Vision and make sure that e.g. camera is not opened when an application is not running. Ignoring Android activity lifecycle would block other applications which would like to use a camera too.

Even though that integration is simplified, activities containing business logic are flooded with code that takes care of *CameraSource* instance. As a consequence, it decreases maintainability of source code.

2.6 Metrics

Previous analysis has shown that existing solutions and frameworks make integration and basic usage too difficult, which could be one of the reasons

Library	Source Code Language	Integration	Lines of Code
OpenCV	C++, Java	Local shared-object libraries, Java wrappers	approx. 330
Mobile Vision API	Java	Gradle Dependency	approx. 530

Table 2.1: Current solutions' comparison.

why eye tracking is not used in mobile applications as much as it could be.

Based on that, following metrics were set:

- *easy integration* - framework should be available via Gradle repository which later requires only one line to download the whole library.
- *separated logic* - a logic of business classes and eye tracker handling should not be mixed.
- *extensibility* - it should be possible to extend framework's logic.
- *Java compatible language* - to prevent integration of C++ libraries, a framework should be written in language that Java can cooperate easily with.

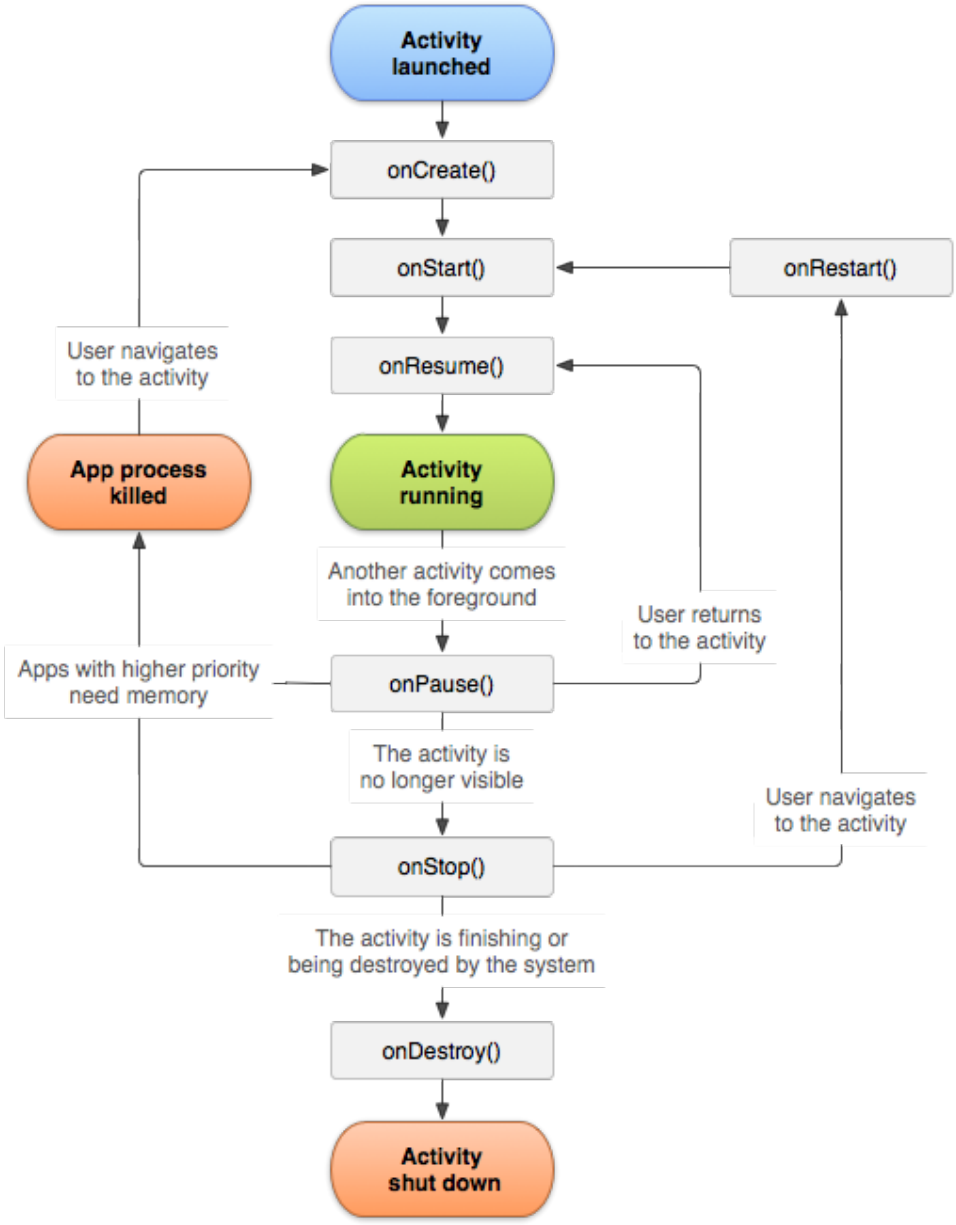

Figure 2.5: Activity Lifecycle. (Undertook from [11])

Chapter 3

Related Work

Using eye tracking to collect passive information from a camera on Android and later adapt UI is not the only use case. This section describes related use cases that are taking advantage of eye tracking.

Tobii Technology [14] is a Swedish company that focuses on developing both eye tracking HW and SW. Tobii's product range contains glasses, screens or tables with integrated eye trackers or standalone eye trackers that are connected with appropriate SW.

Based on eye trackers from Tobii, Microsoft Corporation introduced a new way of controlling Windows 10 OS [15]. The user has to own Tobii tracker that is connected to a computer and user is later able to control the whole system without a need to use a keyboard. The system combines eye tracking with text-to-speech recognition.

Also another company which is for change known for its computer games uses eye tracking - Ubisoft Entertainment. Ubisoft recently released a new game called Assassin's Creed Origins [16]. The game is using Tobii trackers to rotate visible area based on the place where a user is looking. As an example, the user can shoot to a place where he is looking, pick loot by looking at it and much more.

Closely related to eye tracking are face recognition and comparison. Goal of face recognition system is either to identify a person based on the associated record stored in a database - this would be called verification process, or to find out the identity of a person based on all records stored in a database - so-called identification process.

Apple uses facial recognition as a part of its Face ID [17] technology that is used to unlock the phone.

Face comparison is supported by many providers, as an example could be mentioned Amazon Rekognition module which is a part of AWS.

Chapter 4

Analysis

Keeping in mind eye tracking examples from Chapter 3, the framework will support as many features as possible.

It will focus on notifying parent application with regular eye movement updates and together with that, the framework will offer support for dynamic authentication similar to Apple Face ID.

The framework will be designed to meet all metrics from Section 2.6, which means to offer simple integration, low number of lines of code and a widely used programming language.

4.1 Framework's Interactions

Sections that follow depict and describe sequence diagrams related to the framework.

4.1.1 Subscription to the Framework

The diagram in Figure 4.1 shows initialization of eye tracking framework and also a subscription to eye tracking context updates.

When an application is opened, Android OS instantiates *MainActivity* and calls it's lifecycle methods. As a first is invoked *onCreate()* where the framework's database must be initialized. Initialization is done by calling *EyeTracker.init(this)*.

Once the database is initialized, *MainActivity* has to make sure that a new instance of *EyeTrackerFragment* is created and attached.

When necessary, *MainActivity* creates more fragments and these fragments are attached to this activity and call *EyeTrackerAnnotationProcessor.inject()* in their *onAttached* method that registers them to receive context updates.

4.1.2 User Registration

Figure 4.2 depicts user registration to *EyeTrackerFragment*.

When a developer decides that it is right time to register the user, usually on a login screen, the application calls *registerUser()* method with username and *PictureTakenListener* as arguments. *EyeTrackerFragment* takes a picture

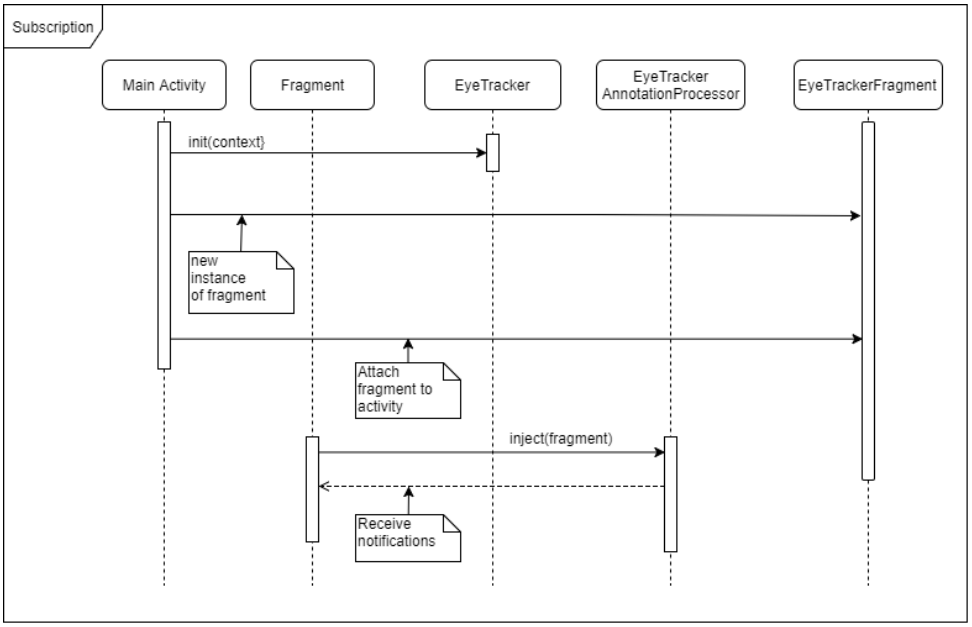

Figure 4.1: Sequence diagram of subscription to fragment.

of a user using a front facing camera, stores the picture on a file system, rotates the picture if necessary and stores picture's path to file in a database.

The picture needs to be rotated because there are mobile cameras that take pictures in a horizontal mode as a default and then rotate it manually. With some mobile brands, a picture is not rotated as it should and it remains in horizontal position.

When registration is completed or an error has occurred, *PictureTakenListener* is notified with result status of registration and bitmap of a picture that was taken if no error was detected.

4.1.3 User Verification

Figure 4.3 depicts user verification using *EyeTrackerFragment*.

To handle dynamic authentication correctly, *MainActivity* should call *verifyUser()* method on *EyeTrackerFragment* and pass username and *UserVerificationListener* as arguments.

EyeTrackerFragment loads a path to a file associated with the username from a database and loads a picture from a file system.

It then takes another picture of a user using a front facing camera and compares both pictures using AWS Rekognition service to check if both pictures contain similar persons.

When a response from AWS Rekognition service is received, it is parsed into *UserVerificationResult* and an application is notified with a result of a comparison via *UserVerificationListener*.

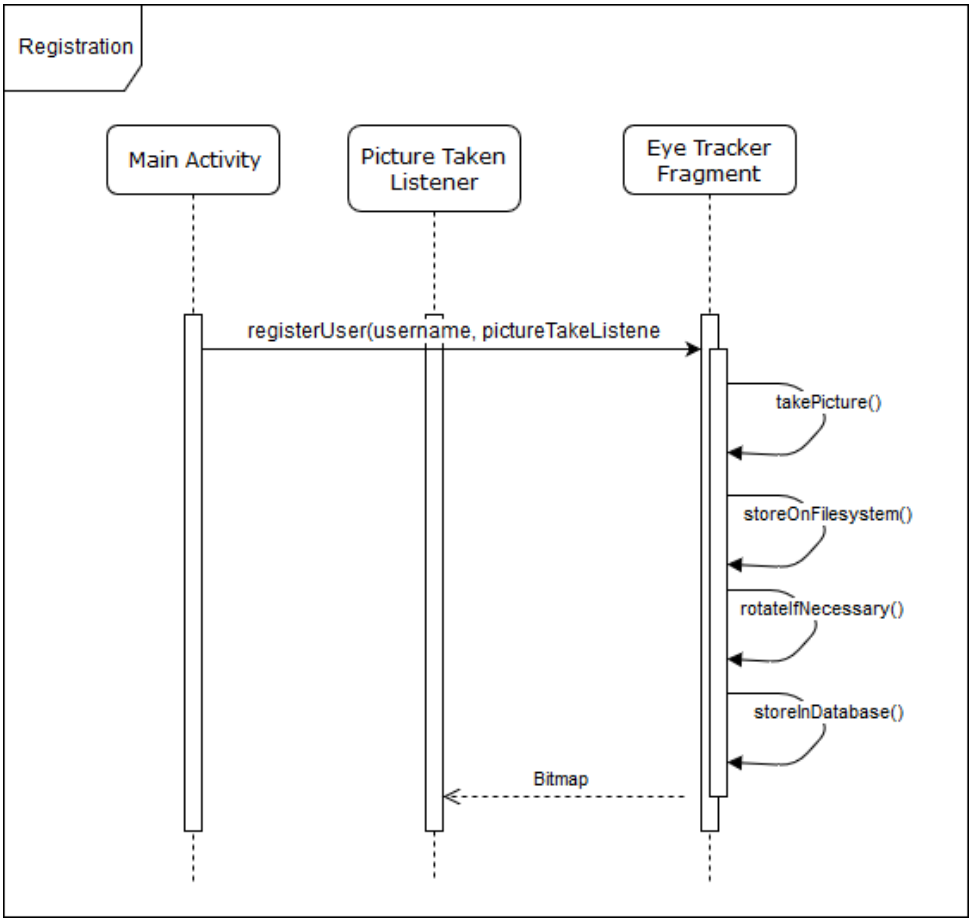

Figure 4.2: Sequence diagram of user registration using EyeTrackerFragment.

4.2 Components of Application

The diagram in Figure 4.4 shows components of the application with underlying Eye Tracker framework.

Host application runs on Android OS. It consists of UI classes and classes that handle business logic. UI classes are encapsulated by Fragments. Fragments are the inner part of *MainActivity*, visualize application's UI and represent reusable components that are subsections of the whole screen.

MainActivity serves as a wrapper for all fragments and communicates with *EyeTrackerFragment* which is a part of Eye Tracker Framework.

Eye Tracker Framework contains *EyeTrackerFragment* and underlying Mobile Vision API framework which tracks user's face and eyes.

Framework depends on AWS SDK which is a component of AWS and provides communication with AWS Rekognition Module running on AWS Cloud. AWS SDK helps to analyze similarity between persons in two different pictures by uploading both images to AWS Rekongition, handling network communication and parsing response data.

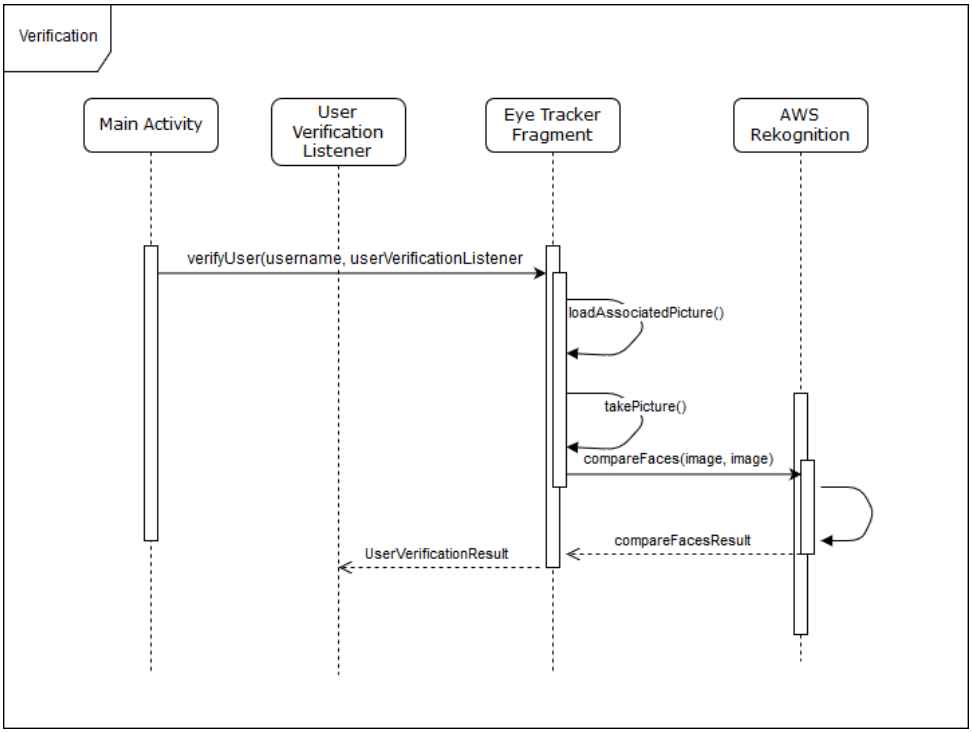

Figure 4.3: Sequence diagram of user verification using EyeTrackerFragment.

4.3 Provided Use Cases

Figure 4.2 depicts all use cases that developer can do with the framework.

Most importantly, the library provides multiple eye tracking events that application can subscribe to. Once these events happen, developer's methods annotated with specified annotations are invoked.

Furthermore, library enables to take advantage of dynamic authentication. It consists of registering and verifying a user. User's photo is captured in a registration process, stored and later queried and compared with a photo taken in a verification process. The developer is then notified via callback method whether user's identity based on facial recognition does or does not match.

A building of user attention tree is necessary for adaptation optimization and it serves as a tool to amend UI appropriately. Once events mentioned above are detected, framework stores events in a database under associated attention section. Attention sections allow the developer to define inheritance between application's sections and once attention tree is built, it queries all child of root section requested by developer and loads associated events for each section.

By data analysis of user attention tree, incorrectly built UI can be analyzed and changed appropriately to do not display unimportant content as a first or similar problems.

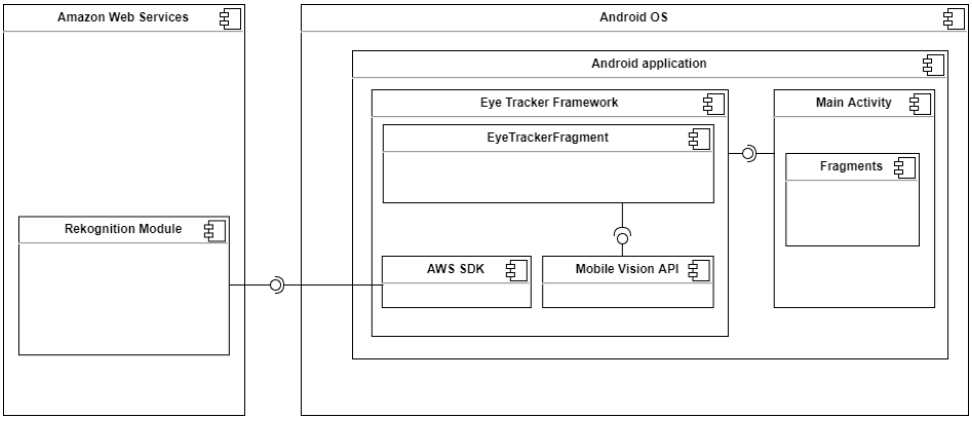

Figure 4.4: Diagram of Components.

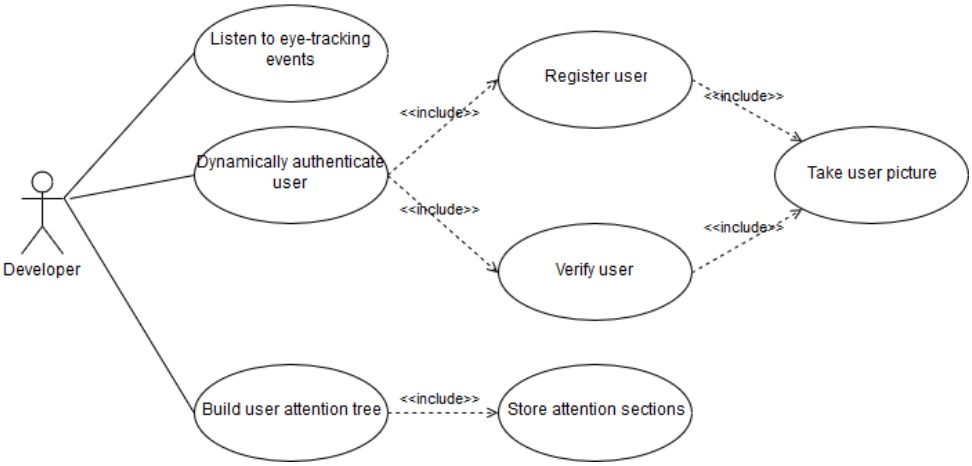

Figure 4.5: Use Case Diagram.

4.4 Framework's Internal Structure

The class diagram on Figure 4.6 visualizes internal structure of the framework. To preserve a reasonable level of complexity, it does not picture all support classes and methods, but only the ones that are exposed to a developer using the framework.

The framework is initialized by Activity. The Activity represents Android Activity mentioned in previous sections. This Activity initializes *EyeTracker* class which is a singleton class that provides a database for the whole framework. Later, Activity attaches *EyeTrackerFragment* as one of its children views. Other fragments subscribe to *EyeTrackerAnnotationProcessor* to receive updates.

EyeTrackerFragment provides functionality for eye tracking itself. It creates *CameraController* that manages *CameraSource* class. *CameraController* is notified by *EyeTrackerFragment* with updates to fragment lifecycle in order to properly manage *CameraSource* which is a part of Mobile Vision API framework and provides access to a camera on a device.

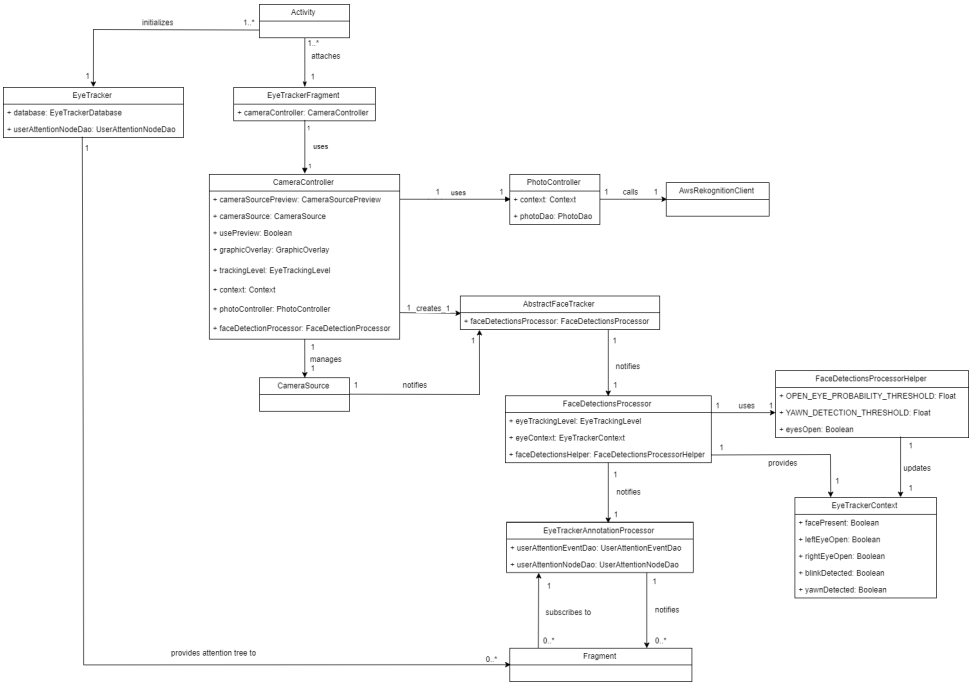

Figure 4.6: Class Diagram.

CameraController creates *CameraSource* and passes a subclass of *AbstractFaceTracker* as an argument. *AbstractFaceTracker* is notified by *CameraSource* with recognition updates and notifies *FaceDetectionsProcessor*.

FaceDetectionsProcessor provides *EyeTrackerContext* and *FaceDetectionsProcessorHelper* updates context based on predefined rules to find context updates. *FaceDetectionsProcessor* then chooses which events should be triggered and calls *EyeTrackerAnnotationProcessor* to which fragments subscribed to receive updates.

EyeTrackerAnnotationProcessor goes through a list of subscribers and notifies each with an event and a context.

As a final step, once there are enough data in the framework, fragments can build attention tree using *EyeTracker* class.

4.5 Database Structure

As mentioned above, framework stores tracking data to be later processed and evaluated.

Figure 4.7 shows the structure of a database.

UserAttentionNode represents a section of a screen or the whole screen that subscribed to receive eye tracking events. It contains id, the id of its parent and section identifier. Section identifier allows to uniquely identify screen of a subsection of a screen in attention tree.

Each *UserAttentionNode* has a parent and can have multiple children. Children are queried by parent id. It also contains a list of *UserAttentionEvents*.

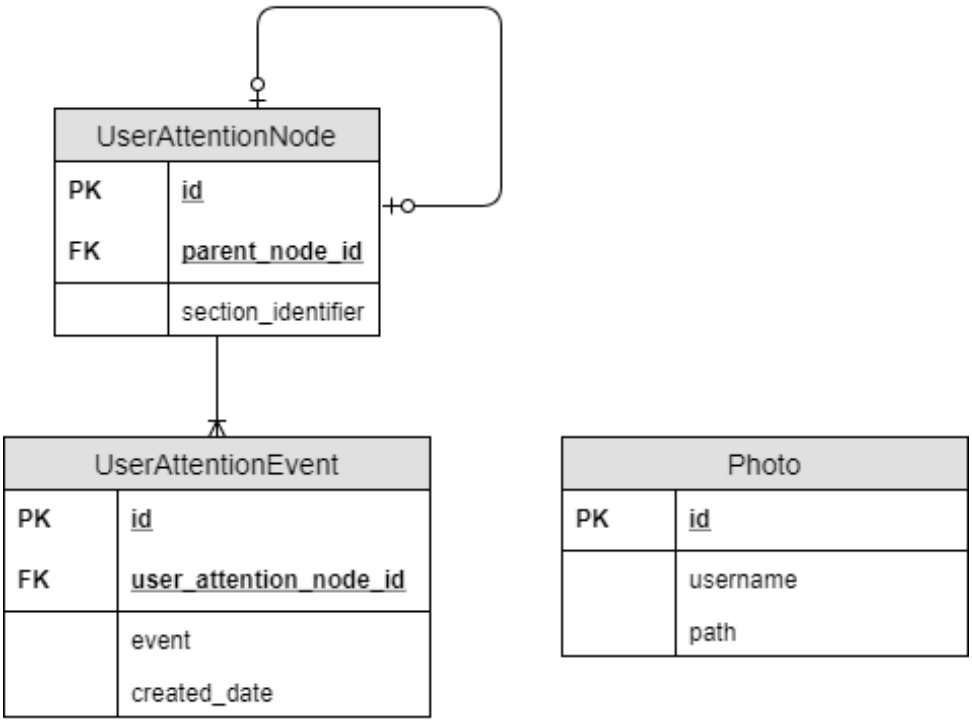

Figure 4.7: Database Structure.

UserAttentionEvent contains id, user attention node id linking back to associated *UserAttentionNode* and in addition to that event name and date when an event was created. The entity represents a single event in tracking process, e.g. that user regained or lost attention. It allows the developer to identify where users are losing attention.

Last entity is *Photo*. It consists only of id, username and path to a file and is used together with dynamic user authentication. When a user is registering in an application, a photo is taken via Eye Tracker Framework, stored on the file system and its path is saved into a database with an associated username.

Once verification process begins, a new photo is taken, the previously captured image is loaded from a file system based on a file path in the database and photos are compared.

Chapter 5

Implementation

The framework is implemented for Android OS using Kotlin programming language that is able to communicate with Java without any changes needed.

Eye Tracker Framework relies on Mobile Vision API which provides a frame processing and AWS Rekognition module which compares faces in uploaded images.

5.1 Platform dependency

The framework was specially designed for Android OS, using libraries targeting this OS and therefore it is not possible to migrate it to another platform.

5.2 Limitations

Unfortunately, a current version of Mobile Vision API does provide only a limited set of information that prevents tracking exact position where a user is looking to. Because of that, it is not possible to build heatmaps, scanpaths or to observe gaze points.

5.3 Project Structure

Eye Tracker project consists of following modules:

- eyetracker - a framework which enables usage of eye tracking and depends on Mobile Vision API and AWS Rekognition.
- sample - application which uses eyetracker module to demonstrate concrete use cases. This module also serves as an example application for user testing.
- sample-with-preview - simple application showing integration and that has enabled preview to see what framework actually tracks.
- sample-dynamic-authentication - application demonstrating implementation of dynamic authentication using Eye Tracker framework and Firebase Authentication to login via Facebook

5.4 Integration

The framework is distributed using Jitpack.io repository. While integrating the library into an existing project, it is necessary to first register Jitpack repository in project build.gradle file as can be seen in Listing 5.1.

Listing 5.1: Project Jitpack Repository Dependency

```
allprojects {
    repositories {
        maven { url 'https://jitpack.io' }
    }
}
```

Once the repository is registered, all that is needed to download the framework is to specify a dependency in module's build.gradle file as shown on Listing 5.2.

Listing 5.2: Gradle Dependency

```
implementation 'com.gitlab.jakubgruber:eye-tracker:1.0.0'
```

After resolving dependencies, *EyeTrackerFragment* can be initialized and attached to parent activity. The fragment is initialized using a builder pattern. See Listing 5.3.

Listing 5.3: Fragment Initialization

```
val eyeTrackerFragment = EyeTrackerFragment.Builder
                    .withFps(20.0f)
                    .withPreview(false)
                    .withPrecision(
                            EyeTrackingLevel.HIGH_PRECISION)
                    .withAwsAccessKeys(
                            accessKeyId,
                            secretAccessKey)
                    .build()

supportFragmentManager.inTransaction {
                add(R.id.fragment_container_eye_tracker,
                eyeTrackerFragment, EyeTrackerFragment.TAG)
            }
```

After performing previous steps, *EyeTrackerFragment* is running in the background of an activity, but no fragments have subscribed to receive eye tracking events. To do so, fragments should have at least one method annotated with one of specified annotations *(@OnContextUpdate, @OnFaceAppeared, @OnFaceLost or @OnYawnDetected)*. Methods must be public and accept argument of type *EyeTrackerContext*. Example method can be seen on Listing 5.4.

Listing 5.4: Method to receive eye tracking updates.

```
@OnFaceAppeared
fun onFaceAppeared(eyeContext: EyeTrackerContext) {
    activity!!.runOnUiThread {
        if (!videoView.isPlaying) {
            videoView.start()
        }
    }
}
```

Once fragment has method matching example above, it must register to receive eye tracking events, which shows Listing 5.5.

Listing 5.5: Registering for eye tracking updates.

```
EyeTrackerAnnotationProcessor.inject(this)
```

At the end of fragment lifecycle, it should unsubscribe from updates as shown in Listing 5.6.

Listing 5.6: Unsubscribe from eye tracking updates.

```
EyeTrackerAnnotationProcessor.disconnect(this)
```

5.4.1 Requirements and recommendations

- There must be only one instance of *EyeTrackerFragment* in an activity.
- An application should be built on fragments or every activity that needs to use eye tracking should handle its own *EyeTrackerFragment*.
- To use user registration and verification, the developer has to request AWS access credentials that provide access to AWS Rekognition. In sample applications, credentials are stored in *src/main/raw/aws.properties* file and are read by *PropertyReader* class. It should also not be staged within any of version control systems.
- An application should adapt only a reasonable amount of UI elements based on eye tracking data. Firstly, because it could get confusing for users and secondly because changing UI with every *EyeTrackerContext* update would have a significant impact on application performance.

5.5 EyeTrackerFragment

Eye Tracker Fragment is a class that provides API between Eye Tracker Framework and a host application. It wraps AWS SDK, Mobile Vision API and synchronizes access to camera according to Android lifecycle. If it is built with calling *.withPreview(true)*, it also displays camera frames with face and eye tracking graphic.

The following section explains the usage of available methods on Eye Tracker Fragment.

5.5.1 registerUser method

This method enables registering of a user with an associated username that is passed as an argument into *EyeTrackerFragment*. As a part of registration process, a new picture of the user is taken by a front facing camera and picture is stored on a file system. A new record is created in the database linking username with captured image on the file system.

PictureTakenListener, passed as an argument, is notified with a captured image in form of Bitmap. The bitmap can be shown to the user to see what picture was stored.

In case that picture would not be of an expected quality (depends on a developer who integrates the framework), it is possible to register user again. All that needs to be done is to call *registerUser* method once more. An old photo is replaced by the new one and is deleted. The old photo cannot be retrieved anymore after new registration.

5.5.2 verifyUser method

The following method is probably the most complicated according to resources. When called, a new picture of a user is taken again. Based on a username passed as an argument, library checks if a user has already registered. If not, no database record is found and verification process is canceled.

In other cases, registration photo is loaded from a file system and both pictures are sent to AWS Rekognition via AWS SDK. AWS Rekognition evaluates both pictures, compares people on those images and notifies AWS SDK back.

Based on the results received from AWS SDK, *UserVerificationListener* passed as an argument is notified with *UserVerificationResult*.

UserVerificationResult acquires VERIFIED value if a face on both pictures matches. NOT_VERIFIED if a face does not match, NO_INFORMATION if a user with given username was not registered yet or UNEXPECTED_ERROR if there is another error, e.g. in network communication.

5.6 Communication between Fragment and Activity

5.6.1 Fragments

It is recommended to have an activity that has its business logic built by fragments. These fragments are attached and handled by parent activity. One of the fragments has to be *EyeTrackerFragment* that notifies other fragments with context updates.

PictureTakenListener

Serves as a callback for *registerUser* method. Receives Bitmap with captured image and *TakePictureResult* which explains possible errors. It can be in state of:

- OK - a picture was taken successfully and a bitmap will not be null.
- NULL_USERNAME - username was null or empty. Could not capture and save a photo. A bitmap will be null.
- UNEXPECTED_ERROR - something unexpected has happened (e.g. IO operation). A bitmap will be null.

UserVerificationListener

A listener that needs to be passed to *verifyUser* method. Is later notified once AWS response is available. Method *onUserVerified* is invoked and *UserVerificationResult* is passed as an argument. It can be in one of following states:

- VERIFIED - user was successfully verified and matches the one that registered before.
- NOT_VERIFIED - a user could not be verified. A user of an application has changed.
- NO_INFORMATION - verification precedes registration. No information is present.
- UNEXPECTED_ERROR - something unexpected has happened (e.g. network communication error, AWS Rekognition unavailable).

5.6.2 EyeTrackerContext

Context class holding information about user's eye updates. Is automatically updated at intervals that were specified by calling *withFps* method on *EyeTrackerFragment.Builder* class.

Class contains following information:

- facePresent - specifies whether user's face is present. It means that he's paying attention to content.
- leftEyeOpen - indicates whether the left eye is opened.
- rightEyeOpen - indicates whether the right eye is opened.
- blinkDetected - indicates that user has blinked.
- yawnDetected - - indicates that user is yawning. A user might be tired.

5.7 EyeTracker

EyeTracker is a singleton class (Kotlin *Object*) which provides initialization of a database and once application runs for long time enough, it returns *UserAttentionTree* data by *buildTree* method.

The method returns an object of type *UserAttentionTree* which serves as a holder for *UserAttentionNode*. This node is the root of the tree and contains its children nodes and events related to the node.

When calling the method, section identifier must be provided as an argument. Section identifier matches sectionIdentifier from *UserAttentionNode* and allows to uniquely tag section of a screen. Example call visible on Listing 5.7:

Listing 5.7: Retrieving UserAttentionTree.

```
val tree = EyeTracker.buildTree("application_homepage")
```

Chapter 6

Eye Tracking Usages

Eye tracking is being used more and more in a wide area of situation and following sections suggest possible use cases with explanation how it eye tracking could be used in given situations.

Some use cases are depicted in figures.

6.1 Automatically pause/resume video

To stop playing video, it is necessary yo tap anywhere on the screen. It does not sound so difficult, but when something happens, e.g. someone is at the door and rings the bell, it is necessary to open quickly and stopping a video starts to be a bit slow.

By integrating eye tracking into an application, it is easy to detect that user is no longer watching the video and stop it. Once he is back, video resumes automatically again.

A prototype of automatic video playback is depicted in Figure 6.1.

6.2 Notify user to regain his attention

Applications are created for a user to simplify some tasks, but as an exchange, it is expected from users to pay attention to what is going on in the application. Therefore a developer wants a user to pay attention to a content as much as possible, detect attention loss and try to regain it.

Attracting of user's attention might be also reversed, e.g. in case of GPS navigation. Mobile devices are commonly used as a replacement for a navigation in cars, but it comes with a risk.

Original navigations are single-purpose devices, but Android phones offer much more.

Because of that, user's are using phones to check emails and other news feed while driving and it is worth considering to integrate eye tracking into map application to detect if a user is not staring at the screen too often and in the case it happens, notify the user to watch the road instead of a phone.

Suggested methods that can be used to gain user's attention are explained in the next subsections.

6.2.1 Change UI elements

If a screen is static, a user does not care about it and pays attention to something else. By increasing a text size, changing color or position of selected elements, attention can be regained.

6.2.2 Increase Brightness

Increasing brightness would be efficient especially at night because it is visible immediately. One thing to keep in mind is that phones are using the highest brightness possible when in sunlight, so it would not work for these cases.

6.2.3 Play Short Sound

A user usually pays attention to sounds that a phone does and by choosing a sound that is unknown to the user, attention can be regained quickly. As a disadvantage stands a fact that choosing of unknown sounds can be ignored as the user does not expect them.

6.3 Full Device Control

Talking about eye tracking, one usually imagines an application that is fully controlled by eye movement and that does everything in a way that it does not require extra user interaction. Referring to Section 5.2, existing frameworks have its limits and this one is one of them.

Development of such applications is very pricey and it is not being implemented after an application is developed, but its designed while keeping eye tracking already in mind.

As an example of such application is EVA Facial Mouse [13] that replaces a keyboard in Android OS to help handicapped persons to control device, as they are not capable of using their hands.

User firstly synchronizes front facing camera with application to be able to accurately track user's face and eyes. After that, he is able to move his head and eye around and navigate within the application.

Following subsections describe elements being used to provide fully eye movement-controlled application.

6.3.1 Blink to switch activities

Speaking of full device control, one aspect of eye tracking that helps is a blinking. It is easy to detect and can be used as one part of navigation in fully eye-controlled devices. The user would just blink and activities would be switched based on a predefined order.

A downside is that blinking is something natural and does not necessarily means that user wants to switch screen, so it can get overused.

It could be however solved by setting a timeout for a blinking duration. Setting a higher timeout would eliminate unexpected blinking.

6.3.2 Blink to scroll

Another possibility to use eye blink is scrolling down while browsing in the Internet or reading a long text. That could be used in applications simulating e-book readers. A lot of users would use it as they lay down on the bed and just read and then, if correctly integrated, the application would be changing pages and scrolling through the content.

6.3.3 Long Gaze

Looking at something for a long time expresses interest. Because of that, long gaze serves as a core block while building applications fully controlled by eye movements.

Long gaze expresses a wish to interact with elements, e.g. to zoom them, click on them or close them.

6.3.4 Midas Touch Problem

Approaches used for full device control by eye tracking must be chosen wisely, mainly because of so-called "Midas Touch Problem". It states that a border between looking at an element to examine it and looking at an element to interact with it is really thin [18].

Users might want just to examine given element, but incorrect implementation would evaluate long gaze as an impulse to trigger defined event, e.g. click on the element.

6.4 Security

Eye tracking and face identification can be easily used to increase application's security.

6.4.1 Hide Secret Content

As was already mentioned in Chapter 1, mobile devices are used almost everywhere and hands in hands with that goes the fact, that it limits privacy of users. It often happens in a public transportation that people stare into a screen of other users.

With growing development of mobile applications, users are migrating their web applications to mobile. One kind of them is a banking application that allows users to manage their finances. It contains details about their financial situation and it is something that users might not want to tell anyone.

While browsing through an application, all the content should be visible, that cannot be changed, but once attention is lost, it makes sense to hide sensitive information in order to prevent unauthorized people to see the content.

Figure 6.2 depicts an example of a fragment in a banking application that automatically hides bank account balance to prevent strangers to see it.

6.4.2 Dynamic User Authentication

Classic authentication into an application requires password or fingerprint input. Password or fingerprint be might secure, but an authentication process is performed just once and as long as the user does not close an application or log out, he is authenticated.

Even though most of the users use lock on their device, their mobile can be stolen in a situation when it is not locked. In such a situation, all application are accessible and owner of the device would usually be also logged in.

An idea of dynamic user identification is that user would register and log in in the beginning, but the application would verify his identity also when running. Example form can be seen in Figure 6.3.

It is understandable that application cannot require password every 5 minutes, but it could take a photo during a registration process and later compare the stored photo with the one that would be taken while a user is using application.

By doing so, the user would not be annoyed by never-ending authentication, but the application would still be automatically locked in case of unauthorized access. As an addition to that, verification using a photo would stop users to reuse their credentials between different accounts.

However, such applications should keep in mind that phone can be borrowed and in that case, it should not be locked automatically.

6.5 User Interface Analysis

Referring back to Subsection 2.3.1, eye tracking is used as a support tool to evaluate UI. Going more in depth, it provides tools to analyze scanpaths, heatmaps, fixations or areas of interest.

When a developer is evaluating and processing results from *UserAttentionTree* (see Section 5.7 for more details), he must keep in mind known properties of correctly designed UI.

By observing events logged by the framework to specified *UserAttentionNodes*, a developer can see patterns in user behavior. When a user loses his attention often in parent section rather than in its children, it makes sense to make UI elements more *findable* and consider switching parent and children nodes to expose desired content higher in a navigation tree. Remembering the *desirable* property of UI, sections that are not visited and that cause user attention loss should be removed from application completely.

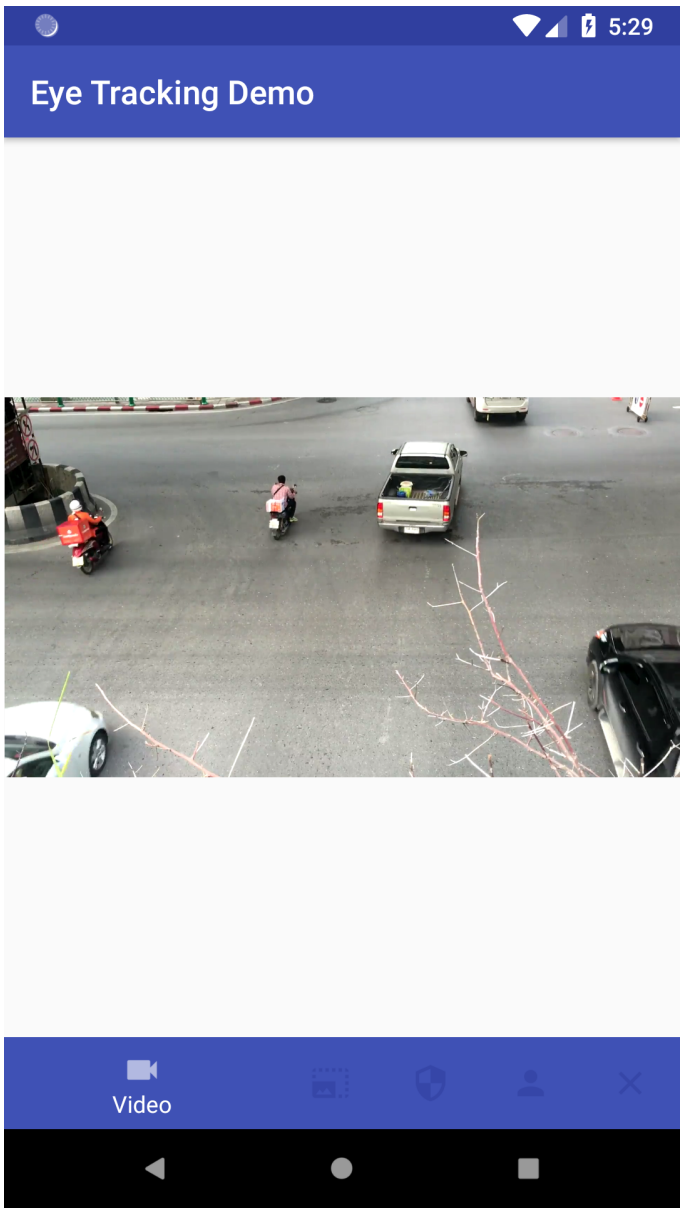

Figure 6.1: Automatically pausing/resuming video

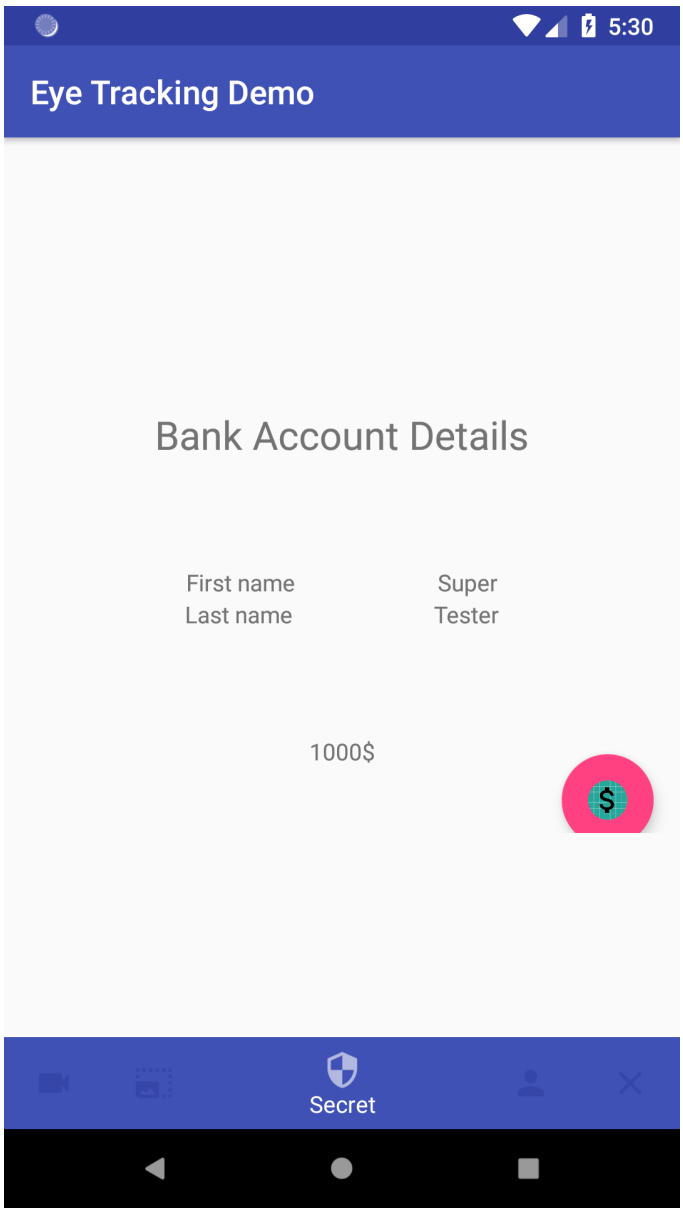

Figure 6.2: Automatically hiding a private section of an application.

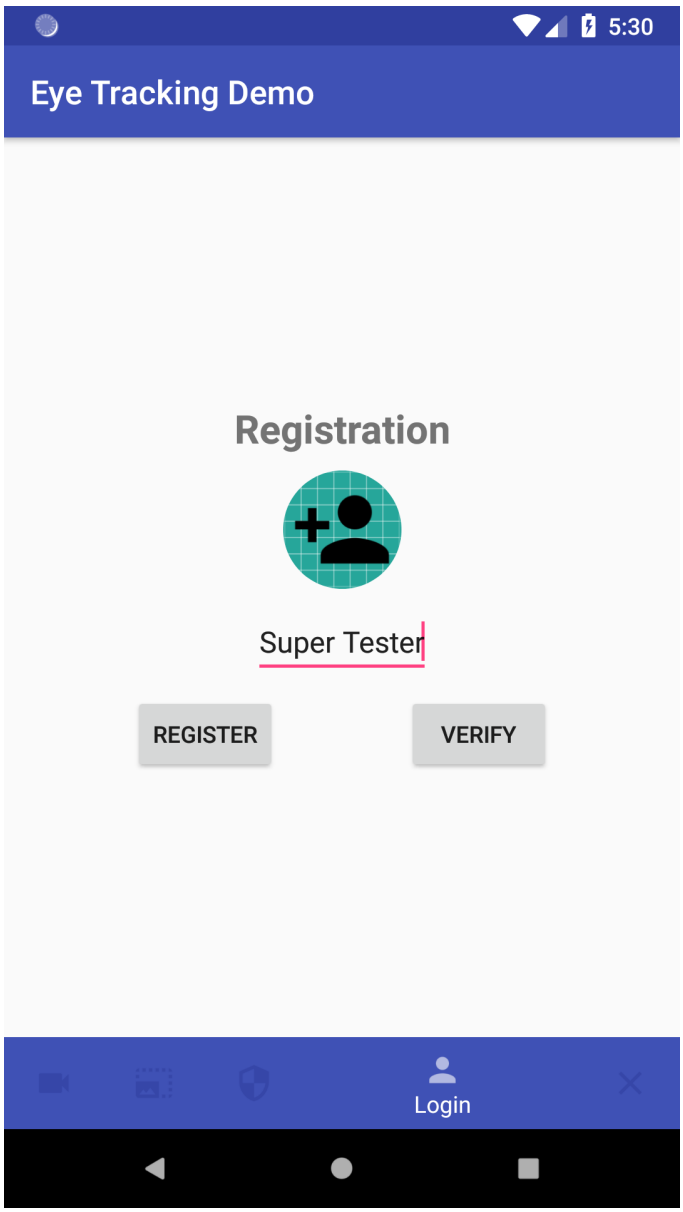

Figure 6.3: Use case showing a use of dynamic authentication.

Chapter 7

Testing

Testing is without a doubt an important process while developing an application. Unfortunately, it is not possible to perform automatic testing of applications that rely on external hardware, especially input from a camera.

Because of that, testing of the application was performed using a user testing of a sample application in combination with user questionnaire.

Testing subjects then filled a questionnaire that consisted of 5 statements dedicated to each section.

7.1 Testing Method and Evaluation

Sample application contained 3 sections, where Eye Tracker Framework was integrated. Each section was separately evaluated by System Usability Scale[1] approach.

SUS concept was amended, but the main concept was preserved. The questionnaire consisted of 3 sections, one for each testing sample. Every section had 5 statements that were evaluated by a user on a scale from 1 to 5, where 1 means "totally disagree" and 5 "totally agree".

7.2 Data Interpretation

Having three sections with 5 questions each, we can number question from 1 to 15. Positive questions were evaluated by 4 points to "totally agree" going down to 0 to "totally disagree". Negative questions were evaluated oppositely. Table 7.1 contains an overview of how many points were awarded to each question based on answers.

SUS uses 10 questions that are awarded points from 0 to 4 and a sum of points is multiplied by 2,5 to create a scale from 0 to 100.

As our questionnaire contains 5 question, the sum will be multiplied by 5.

Furthermore, SUS states that system awarded 68 and more points can be considered as "above average". The threshold of 68 points will be preserved and considered as a threshold of success.

[1] https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Answer	Points (for questions 1, 3, 6, 8, 10, 11, 13, 15)	Points (for questions 2, 4, 5, 7, 9, 12, 14)
Totally agree	4	0
Agree	3	1
I do not know	2	2
Disagree	1	3
Totally disagree	0	4

Table 7.1: Questions evaluation.

7.3 Video Control Test

A first section was dedicated to controlling a video using eye movements. This use case belongs to a category of active AUI, where a user directly controls application by executed actions. The sample contained a simple screen with a single video that was playing in a loop.

In the test, a user was supposed to start, stop and resume the video only by using eyes, blinking or simulating a complete attention loss by looking away from the screen.

7.3.1 Questions

1. I would prefer to control video by eye movements rather than by finger taps.
2. I think it was more difficult to control the application by eyes.
3. I would image that most people would learn to use the application quickly.
4. I think there was too much inconsistency in the system.
5. I needed to learn a lot before understanding how to control the application.

7.3.2 Answers and Results

Table 7.2 depicts answers of testing subjects according to testing of video control.

Average of points awarded to this sample is equal to **57,5** which is slightly below average set by SUS method which means that users were not really satisfied with the way how video control worked. It may be caused by the fact that video was controlled by blinking which, as explained earlier, might be too sensitive. It can be solved by setting a threshold for blinking detection.

There was also a high inconsistency in results whether it was more or less difficult to control an application. Despite that, users have agreed that most of the people would learn to use this feature quickly.

Q1	Q2	Q3	Q4	Q5	Sum	Sum*5
3	0	2	2	4	11	55
3	1	2	4	3	13	65
0	0	2	3	2	7	35
2	1	3	2	3	11	55
2	2	3	3	3	13	65
4	2	2	2	0	10	50
2	4	4	1	3	14	70
3	3	4	3	0	13	65

Table 7.2: Video Testing Results.

7.4 Secret Hiding Test

The second task in user testing consisted of an example of a screen of a banking application with user private information such as name, email and mainly account balance. Screen fragment was implemented in a way that it was automatically hiding bank account balance in case that user's attention was lost. This example belongs to a group of passive UI that adapts itself without requiring a user to execute an action.

While testing, a testing subject had to observe this single screen, could increase bank account balance to simulate regular usage of a banking application and bank account balance was hiding automatically.

7.4.1 Questions

1. I think I would like my banking application to automatically hide private sections.
2. I feel confused by elements that are automatically hiding.
3. I would feel more comfortable if my private information would be automatically hidden.
4. I found the system very cumbersome to use.
5. I found this function in the system well integrated.

7.4.2 Answers and Results

The data in Table 7.3 indicates that testing subjects were more satisfied with secret hiding than with video control. Its average is **78,75** which is above SUS threshold.

Successful results may be influenced by increasing worries about users' privacy. As can be seen in results of question 6, almost all users would like their banking applications to automatically hide secret information and as question 10 states, the feature was well integrated and easy to use.

Q6	Q7	Q8	Q9	Q10	Sum	Sum * 5
4	3	4	3	2	16	80
4	4	3	4	4	19	95
4	4	4	4	4	20	100
3	3	3	2	3	14	70
1	1	2	2	2	8	40
4	0	4	3	4	15	75
4	3	3	3	2	15	75
4	4	4	4	3	19	95

Table 7.3: Secret Hiding Results.

7.5 Dynamic Authentication Testing

A last task of testing represented a login screen with dynamic authentication integrated using a developed framework. Fragment's UI consisted of input for username and two buttons - one for registration, the second one for verification.

A testing objective was to register into an application by choosing a username and then clicking on a registration button. User's picture was taken and shown to a user. In case of dissatisfaction, a testing subject had a possibility to repeat registration.

Once registration was completed, another objective was to verify self by clicking "verify" button. This simulated dynamic authentication that would be performed automatically by an application at chosen intervals.

Testing subject's goal was to observe success rate of the framework in verification.

7.5.1 Questions

1. I think I would like to use photo capture instead of remembering a password.
2. I do not feel comfortable being photographed often by an application.
3. I feel safer when using dynamic photo authentication.
4. I think that dynamic authentication is unnecessarily complex.
5. I think that it would help me to stop reusing similar passwords.

7.5.2 Answers and Results

As seen in Table 7.4, the final score of the feature was rated by **68,13** points. It is equal to average set by SUS and means that users' feelings very mixed according to this feature.

Looking at single results, high grades in question 11 denotes that most of the testing subjects would like to use dynamic authentication instead of using a password and together with question 15, users have agreed that it would help them to stop reusing similar password between multiple accounts.

However, the test also indicates that some of the subjects felt uncomfortable about being photographed constantly by an application and are afraid of breaking into their privacy.

Q11	Q12	Q13	Q14	Q15	Sum	Sum * 5
4	4	4	3	4	19	95
4	2	2	4	4	16	80
4	3	2	4	4	17	85
1	1	2	2	2	8	40
3	1	1	3	3	11	55
4	0	4	0	4	12	60
2	2	2	2	2	10	50
4	2	4	3	3	16	80

Table 7.4: Dynamic Authentication Results.

7.6 Known bugs

The user testing was very helpful because it consisted of testing the sample application on a variety of different devices.

It has shown up that the framework crashes while running on devices with a processor architecture x86 and Android Marshmallow or higher. This problem is caused by one of the libraries and needs to be solved on a side of a provider.

An error is caused by the fact that the library is text relocations that are not allowed since Android 6.

Chapter 8

Installation

Demo application should be able to run under any version of Android that is equal or higher to 4.0.3. A device must have an access to Google Play Services that are required for additional data that are necessary for Mobile Vision API. Phone must have a front facing camera.

The application was installed using an Android Studio 3.0 running on Windows 10. It was tested smartphone from Sony running Android 8.0.0.

1. Insert CD into CD Rom mechanic
2. Extract eye_tracker.zip
3. Copy extracted folder on disk
4. Open Android Studio
5. Import extracted project into Android Studio *(File -> New -> Import Project)*

Chapter 9

Conclusion

The task of the thesis was to investigate current solutions in eye tracking field and evaluate its advantages and disadvantages. Based on the observed information, set up metrics for a framework, implement the framework, create a sample application and evaluate eye tracking use cases by user testing.

Existing solutions were described in Section 2.5 and metrics for a framework that had to be developed were set. Based on main disadvantages of current libraries, it was decided that the solution created in this thesis must be easy to integrate, implemented in a commonly used programming language by most of the Android developers and its design must be focused on a support for AUI.

The developed framework was therefore implemented in Kotlin language, distributed using a Gradle repository and its integration into an existing project is done by using annotated methods. As seen in Section 5.4, the whole integration is thanks to a chosen design reduced to approximately 20 lines of code necessary to start using the framework. This number is in comparison with OpenCV by 300 lines of code less and in case of pure Mobile Vision API, it is by 500 lines of code less.

Chapter 6 suggested a common approach and use cases how to use the framework to develop an application with AUI and what advantages it brings. Keeping these use case in mind, sample application was developed and tested by users.

According to results of the user testing, it was shown that users found that too difficult to control a video by their eye movements and that there were inconsistencies in the way how the feature was implemented. Despite that, users have expressed themselves that they would like to control the video by eye movements anyway which indicates an increasing potential for eye tracking.

The testing has also proven that more and more users are concerned about their privacy nowadays and because of that, their opinions on a hiding secret information in an application were predominantly positive.

The last task in the user testing pointed out that eye tracking can increase applications' security by replacing a standard way of a registration into a user account by using a dynamic authentication. Testing subjects have also agreed that doing so would decrease the number of passwords they reuse in

different applications.

To summarize the thesis overall, it was shown that eye tracking area of study is still missing a framework that would fully support a combination of eye tracking and adaptive contextual applications. Such a framework was developed as a part of this thesis and it was proven by the SUS study that users were reacting mostly positively to presented samples. The score of all samples was around a specified threshold by the SUS and therefore the whole thesis can be considered as a success.

Chapter 10
Future Work

Looking forward, as a first will be fixed a bug detected during user testing that causes an application to crash on some devices. The fix will require replacing a current dependency on AWS SDK for a different one that also offers a face comparison.

In addition to that, the framework will be extended by more features, mainly focused on a UI that is fully-controlled by eye movements. Due to limitations of Mobile Vision API that currently does not offer all properties that are necessary to calculate exact position where a user is looking to, OpenCV will be used instead. The framework was designed with a focus on an easy replacement of an underlying framework that provides eye tracking data and therefore framework's API will not be influenced.

Going even further, the framework will be used in a combination with Adaptive Application Structure (AAS)[2] framework to observe patterns in a user behavior reflecting both eye tracking movements and the way how a user browses through an application. The AAS framework builds a tree from an application's navigation menu and tries to restructure the menu based on the numbers of visits in each node.

To support a detection of patterns in user behavior, a server with REST API will be implemented. The server will evaluate eye tracking and adaptive application structure data by machine learning and find recurring patterns. Using a database of detected patterns, the server will offer recommendations to a developer how to build the most effective UI.

10.1 Patterns to Evaluate

A combination of the eye tracker framework and the AAS framework will result in an application that demonstrates a usage of both frameworks together. The application will be tested by scenarios that follow.

Once the underlying framework will be replaced, it will be possible to construct heatmaps and scanpaths. Both heatmaps and scanpaths will serve as a comparison to the tree that is built by the AAS framework.

Analyzing the heatmap, the final framework would be able to distinguish the importance of the elements not just by calculating the number of visits, but also by an analysis of the time that a user dedicates to inspect every single

element. Such a comparison should increase the accuracy of an adaptation that the AAS framework performs after a large enough dataset is collected. Moreover, speaking about an amount of data, a usage of eye tracking would speed up an adaptation, because eyes provide meaningful information much faster.

A common problem that the AAS framework deals with is a situation where a *parent has too many elements* below. An example for this is an application that displays many sections of the same type, e.g. articles in a news application. It is expected that a user's attention will not be influenced by the structure, but by the content of each article. This experiment would be simulated by creating two or more applications that have an exactly same structure, but the content of each application would be different. In the first case, a content that should be interesting for a user would be chosen and in other cases, the application would display something unimportant.

Doing so, the experiment should prove that the attention and usability of the application are not influenced just by correctly built structure, but also by a provided content. If this would be confirmed, the framework would be then tested in a combination with a framework that can adapt the content based on a user's needs, e.g. by inspecting the text. With every iteration of integration a new adaptation framework, the UI should improve to attract user's attention more and more.

Another usual problem in adaptive application structure rises with *a navigation tree that is too deep*, which means that every element contains another nested element. Expectations for this case would be that a user will pay attention in the beginning, then lose the attention as he browses through sections in the middle, and finally regain the attention once he gets to desired content. The framework should indicate that sections in the middle are unnecessary and restructure a navigation in an application. The attention lost in the middle of browsing should be noticeable when inspecting scanpaths and the time that a user needs to read sections in the middle. The user is expected not to look on the middle sections for too long.

The testing of a combination of both frameworks should produce a general machine learning model that would be later applied to a new data from other applications. If the model would be trained successfully, the UI recommendation for developers should help them to build a UI that would be focused mainly on a gaining of the user's attention while keeping the application's structure simple.

Appendix A

Bibliography

[1] GHAOUI, Claude. *Encyclopedia of human computer interaction.* Hershey PA: Idea Group Reference, c2006. ISBN 9781591405627.

[2] ŠEBEK, Jiří a Karel RICHTA. *Usage of Aspect-Oriented programming in Adaptive Application Structure: New Trends in Databases and Information Systems.* DOI: : ADBIS 2016 Short Papers and Workshops, BigDap, DCSA, DC, Prague, Czech Republic, August 28-31, 2016, Proceedings.

[3] RAHEEL, Saeed. *Improving the user experience using an intelligent Adaptive User Interface in mobile applications.* In: 2016 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET) [online]. IEEE, 2016, 2016, s. 64-68 [cit. 2018-01-13]. DOI: 10.1109/IMCET.2016.7777428. ISBN 978-1-5090-5281-3. Available on: http://ieeexplore.ieee.org/document/7777428/

[4] LIN, Liannan, Wenda QIN a Chuan LONG. *The analysis and practice of the human-computer interaction course system in Stanford University.* In: 2016 11th International Conference on Computer Science & Education (ICCSE) [online]. IEEE, 2016, 2016, s. 865-870 [cit. 2018-01-13]. DOI: 10.1109/ICCSE.2016.7581695. ISBN 978-1-5090-2218-2. Available on: http://ieeexplore.ieee.org/document/7581695/

[5] CHANDRIKA, K R, J AMUDHA a Sithu D SUDARSAN. *Recognizing eye tracking traits for source code review.* In: 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) [online]. IEEE, 2017, 2017, s. 1-8 [cit. 2018-01-13]. DOI: 10.1109/ETFA.2017.8247637. ISBN 978-1-5090-6505-9. Available on: http://ieeexplore.ieee.org/document/8247637/

[6] ZHANG, Wei, Bo CHENG a Yingzi LIN. *Driver drowsiness recognition based on computer vision technology.* Tsinghua Science and Technology [online]. 2012, 17(3), 354-362 [cit. 2018-01-13]. DOI: 10.1109/TST.2012.6216768. ISSN 1007-0214. Available on: http://ieeexplore.ieee.org/document/6216768/

[7] *User Experience Basics* [online]. [cit. 2018-01-11]. Available on: https://www.usability.gov/what-and-why/user-experience.html

[8] *Video Eye Tracker* [online]. 2018 [cit. 2018-01-12]. Available on: https://www.bradford.ac.uk/research/rkt-centres/visual-computing/facilities/eye-tracking/

[9] *Smartphone OS Market Share* [online]. [cit. 2018-01-12]. Available on: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems

[10] *Android version market share* [online]. [cit. 2018-01-12]. Available on: https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/

[11] *Activity Lifecycle* [online]. [cit. 2018-01-13]. Available on: https://developer.android.com/guide/components/activities/activity-lifecycle.html

[12] *Basic Eye Anatomy* [online]. [cit. 2018-01-13]. Available on: http://www.eyesightresearch.org/background.htm

[13] *EVA Facial Mouse* [online]. [cit. 2018-01-13]. Available on: https://play.google.com/store/apps/details?id=com.crea_si.eviacam.service

[14] *Tobii* [online]. [cit. 2018-01-13]. Available on: https://www.tobii.com/

[15] *Windows 10 Eye Tracking* [online]. [cit. 2018-01-13]. Available on: https://support.microsoft.com/en-us/help/4043921/windows-10-get-started-eye-control

[16] *Ubisoft Gaming* [online]. [cit. 2018-01-13]. Available on: https://assassinscreed.ubisoft.com/game/en-gb/news/detail.aspx?c=tcm:154-305094-16&ct=tcm:154-76770-32

[17] *Apple Face ID* [online]. [cit. 2018-04-14]. Available on: https://support.apple.com/en-us/HT208108

[18] *Midas Touch Problem in Eye Tracking* [online]. [cit. 2018-04-19]. Available on: http://longqian.me/2017/01/05/midas-touch/

[19] *Guide to brain-computer music interfacing.* New York: Springer, 2014. ISBN 978-1-4471-6583-5.

Appendix B

Listings

Appendix C

List of Abbreviations

HCI	Human-Computer Interaction
UI	User Interface
AUI	Adaptive User Interface
SW	Software
HW	Hardware
SDK	Software Development Kit
OS	Operating System
AWS	Amazon Web Services
API	Application Programming Interface
BCI	Brain-computer interface
SUS	System Usability Scale
PDF	Portable Document Format
AAS	Adaptive Application Structure

Appendix D

Content of Attached CD

The content of CD is divided into following directories and files:

- eye_tracking_in_adaptive_contextual_applications.pdf - the bachelor thesis in PDF format
- thesis_sources.zip - TeX sources of the thesis
- eye_tracker.zip - an Android Studio project with the framework and sample modules
- attachments.zip - a directory with diagrams and pictures used in the thesis