**Abstract**

Traditional gaze tracking systems rely on either contact and invasive hardware, or ex- pensive and non-standard hardware. To address this problem research has been done into systems that use only simple hardware to create gaze tracking systems. This system attempts to prove that is possible to create a gaze tracking system using a regular web camera and the free, open source Computer Vision library OpenCV. This is achieved by researching various techniques required for such a system and then measuring the performance of the created system. Analysis of the results concluded that while the error of the system was greater than desired, in conjunction with the research done in this field, it is possible to create a robust system using simple hardware.

[**Table of Figures ix**](#_Toc426616311)

[**Table of Use Cases x**](#_Toc426616312)

[**CHAPTER 1 1**](#_Toc426616313)

[**1. INTRODUCTION 1**](#_Toc426616314)

[**1.1 Introduction 1**](#_Toc426616315)

[**1.2 Problem Statement 1**](#_Toc426616316)

[**1.3 Solution 1**](#_Toc426616317)

[**1.4 Product Scope 1**](#_Toc426616318)

[**1.5 Product Perspective 2**](#_Toc426616319)

[**1.6 Product Features 2**](#_Toc426616320)

[**CHAPTER 2 3**](#_Toc426616323)

[**2. LITERATURE REVIEW 3**](#_Toc426616324)

[**2.1 Introduction 3**](#_Toc426616325)

[**2.2 Open Cv Documentation 3**](#_Toc426616326)

[**2.3 Research paper(Gaze Tracking using a reguler web cam 4**](#_Toc426616327)

**CHAPTER 1**

**1.1 Introduction:**

The idea is that for UI/UX testing, it's important to know what parts of the screen the user interacts with. There are heat maps in the browser/desktop world but on smartphones, there are no "mouse pointers" to track. The user only interacts with the screen by touch events and by looking at the screen. The proposed project will aim to "track the gaze" and of the user -- i.e. where are they looking at on the screen -- to decide which part of the screen content interests them the most. This will be for Android native applications and the end product will be a library that can be included in any app to serve the same purpose as heatmaps on desktops. For gaze tracking, we will use OpenCV's port to Android. We will also give a sample android application for testing our library.

**1.2 Purpose Statement:**

This report is to analyze the feasibility of our project from different aspects. For that we have to study about the technology that we require for our project development and hardware required for it. After analyzing the feasibility aspects we have to come with the effective and feasible options that should be adopted in completing the project.

**Problem Statement:**

Basic problems which influenced this project are as follows:

* The idea is, for UI/UX testing, it's important to know what parts of the screen the user usually interacts with.
* There are heat maps in the browser/desktop world but on smartphones, there are no "mouse pointers" to track.
* The user only interacts with the screen by touch events and by glancing at the screen.

**1.3 Solution**

* We are going to make a library for Android based smartphones.
* The library will serve as a tool for UI/UX testing.
* The purpose of the project is to automate the UI/UX testing criteria.
* And this library would be tested by a sample Android application**.**

**1.4 Scope:**

* Automate the UI/UX testing phenomena.
* Target domain is Android.
* End product will be a library.
* Library can be included in any android application for testing.
* An application for testing our library.

The detailed versions of the architectural design, data design, procedural design, design  
constraints and development schedule which are stated in the initial design description  
document will be covered in this document. The decomposed components are covered and  
identified more detailed. Besides this, dependencies are clarified and stated one by one.  
Description of each module including type, purpose, functions and subordinates, interfaces,  
processing and data are given clearly and more detailed with respect to the ones in the initial  
design description document. Also, the user interface will be covered.

**1.5 Product Functions:**

* The library would help the users for UI/UX testing.
* The software would recommend the restaurant to the user according to his choices and past user experience.

**1.6 Produce Features**

**Reliability:**

CMPGR will be designed as a whole simulation of a mobile device. Since mobile devices are one of the most indispensable objects for people in their daily life, our system have to be designed with minimum faults. It will contain a dataset of various hand features with different types and colors in order to decrease the error tolerance of the system. False errors, which are the most dangerous errors for a system to work correctly, will not be occurred in the system.

**Availability:**

When the project will be completed than the intentions are to take a domain and upload it also upload a window phone version on the store so people can use this app to get the recommendations

**Low-cost:**  
Implementation of the stereovision technology by using one low cost cameras provides an inexpensive solution with respect to the current technologies which are already available on the mobile device market.

**Maintainability:**  
Any error occurred while the system is in use will be tolerated and system recovers itself and continues properly.

**Performance:**  
Since the system will work on real time, performance is one of the most important topics of the system. Performance of the system will be high enough that user can use the system without noticeable delays or performance problems.

**Time:**  
Software will work on real time. Response of the system after the user makes an action will be fast enough that does not cause a problem.

**Security:**  
Since the system is compatible for only Unix based OS, it will be more secure for the external attacks.

**Conclusion:**

The conclusion is that we created a library that can be included in any app to serve the same purpose as heat maps on desktops. This will be for Android native applications.

**Use Cases:**

## Face Detection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** |  | | | |
| **Use Case Name:** | Detect Face | | | |
| **Created By:** |  | | **Last Updated By:** |  |
| **Date Created:** |  | | **Last Revision Date:** |  |
| **Actors:** | | User/ App tester | | |
| **Description:** | |  | | |
| **Trigger:** | | When the application is started | | |
| **Preconditions:** | | 1. The user has integrated the UI/UX library with the application 2. The application has started 3. The camera is in working state | | |
| **Post conditions:** | | 1. The face of the user would be extracted through frames generated by the camera. | | |
| **Normal Flow:** | | 1. The user starts the application 2. The camera starts in the backend 3. The camera starts generating frames 4. The frames are sent to open cv port to detect face 5. The face is detected and a rectangle box is placed around it | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | |  | | |
| **Exceptions:** | | 1. The camera fails to start 2. The library doesn’t load | | |
| **Includes:** | | 1. Android cell phone with front camera | | |
| **Frequency of Use:** | |  | | |
| **Special Requirements:** | |  | | |
| **Assumptions:** | |  | | |
| **Notes and Issues:** | |  | | |

## Eyes Detection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** |  | | | |
| **Use Case Name:** | Detect Eyes | | | |
| **Created By:** |  | | **Last Updated By:** |  |
| **Date Created:** |  | | **Last Revision Date:** |  |
| **Actors:** | | User/ App tester | | |
| **Description:** | |  | | |
| **Trigger:** | | When the library has detected the face | | |
| **Preconditions:** | | 1. The user has integrated the UI/UX library with the application 2. The application has started 3. The camera is in working state 4. The face is detected by the library | | |
| **Post conditions:** | | 1. The eyes of the user would be detected. | | |
| **Normal Flow:** | | 1. The user starts the application 2. The camera starts in the backend 3. The camera starts generating frames 4. The frames are sent to open cv port to detect face 5. The face is detected and a rectangle box is placed around it 6. The face is taken as input by the library and then in return it detects eyes from it. | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | |  | | |
| **Exceptions:** | | 1. The camera malfunctions 2. The library stop working | | |
| **Includes:** | | 1. Android cell phone with front camera | | |
| **Frequency of Use:** | |  | | |
| **Special Requirements:** | |  | | |
| **Assumptions:** | |  | | |
| **Notes and Issues:** | |  | | |

## Pupil Detection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** |  | | | |
| **Use Case Name:** | Detect pupil | | | |
| **Created By:** |  | | **Last Updated By:** |  |
| **Date Created:** |  | | **Last Revision Date:** |  |
| **Actors:** | | User/ App tester | | |
| **Description:** | |  | | |
| **Trigger:** | | When the library has detected the eyes | | |
| **Preconditions:** | | 1. The user has integrated the UI/UX library with the application 2. The application has started 3. The camera is in working state 4. The face is detected by the library 5. The eyes are detected by the library | | |
| **Post conditions:** | | 1. The pupil of the user would be detected. | | |
| **Normal Flow:** | | 1. The user starts the application 2. The camera starts in the backend 3. The camera starts generating frames 4. The frames are sent to open cv port to detect face 5. The face is detected and a rectangle box is placed around it 6. The face is taken as input to open CV port which in return gives us location of eyes. 7. The eyes are then taken as input to open CV port which in return gives us location of pupil | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | |  | | |
| **Exceptions:** | | 1. The camera malfunctions 2. The library stop working | | |
| **Includes:** | | 1. Android cell phone with front camera | | |
| **Frequency of Use:** | |  | | |
| **Special Requirements:** | |  | | |
| **Assumptions:** | |  | | |
| **Notes and Issues:** | |  | | |

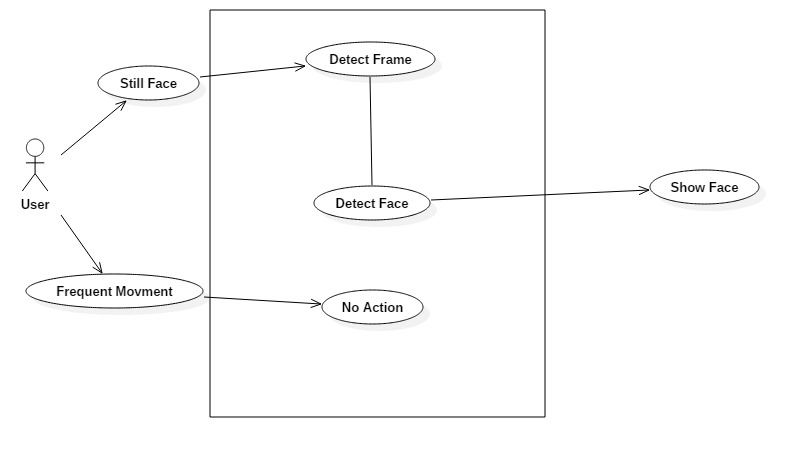
## Corner Detection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** |  | | | |
| **Use Case Name:** | Detect pupil | | | |
| **Created By:** |  | | **Last Updated By:** |  |
| **Date Created:** |  | | **Last Revision Date:** |  |
| **Actors:** | | User/ App tester | | |
| **Description:** | |  | | |
| **Trigger:** | | When the library has detected the eyes | | |
| **Preconditions:** | | 1. The user has integrated the UI/UX library with the application 2. The application has started 3. The camera is in working state 4. The face is detected by the library 5. The eyes are detected by the library 6. The pupil is detected by the library | | |
| **Post conditions:** | | 1. The pupil of the user would be detected. | | |
| **Normal Flow:** | | 1. The user starts the application 2. The camera starts in the backend 3. The camera starts generating frames 4. The frames are sent to open cv port to detect face 5. The face is detected and a rectangle box is placed around it 6. The face is taken as input to open CV port which in return gives us location of eyes. 7. The eyes are then taken as input to open CV port which in return gives us location of pupil 8. The pupil of the person is given as input to the open cv port and in return it’s give the corners. | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | |  | | |
| **Exceptions:** | | 1. The camera malfunctions 2. The library stop working | | |
| **Includes:** | | 1. Android cell phone with front camera | | |
| **Frequency of Use:** | |  | | |
| **Special Requirements:** | |  | | |
| **Assumptions:** | |  | | |
| **Notes and Issues:** | |  | | |

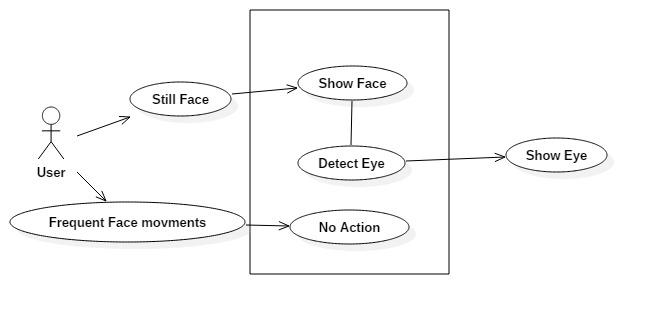
## Heat Map Generation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID:** |  | | | |
| **Use Case Name:** | Detect pupil | | | |
| **Created By:** |  | | **Last Updated By:** |  |
| **Date Created:** |  | | **Last Revision Date:** |  |
| **Actors:** | | User/ App tester | | |
| **Description:** | |  | | |
| **Trigger:** | | When the library has detected the eyes | | |
| **Preconditions:** | | 1. The user has integrated the UI/UX library with the application 2. The application has started 3. The camera is in working state 4. The face is detected by the library 5. The eyes are detected by the library 6. The pupil is detected by the library 7. The corners are detected by the library | | |
| **Post conditions:** | | 1. The pupil of the user would be detected. | | |
| **Normal Flow:** | | 1. The user starts the application 2. The camera starts in the backend 3. The camera starts generating frames 4. The frames are sent to open cv port to detect face 5. The face is detected and a rectangle box is placed around it 6. The face is taken as input to open CV port which in return gives us location of eyes. 7. The eyes are then taken as input to open CV port which in return gives us location of pupil 8. The pupil of the person is given as input to the open CV port and in return it’s give the corners. 9. The corners and computed data later help generate heat maps | | |
| **Alternative Flows:**  **[Alternative Flow 1 – Not in Network]** | |  | | |
| **Exceptions:** | | 1. The camera malfunctions 2. The library stop working | | |
| **Includes:** | | 1. Android cell phone with front camera | | |
| **Frequency of Use:** | |  | | |
| **Special Requirements:** | |  | | |
| **Assumptions:** | |  | | |
| **Notes and Issues:** | |  | | |

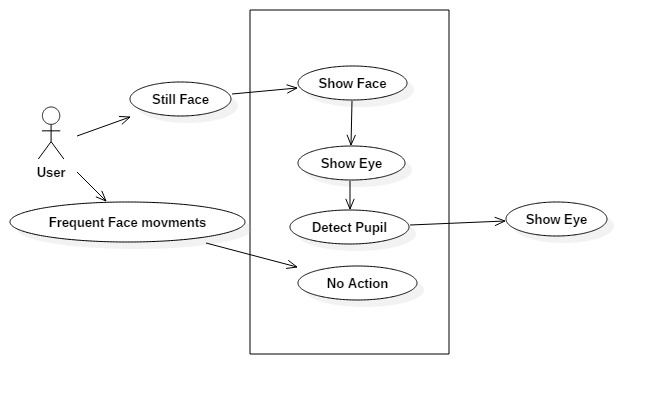
**Face Detection:**



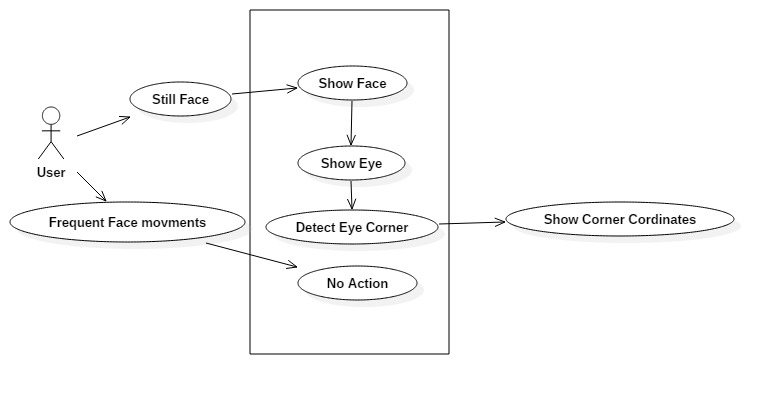
**Eye Detection:**



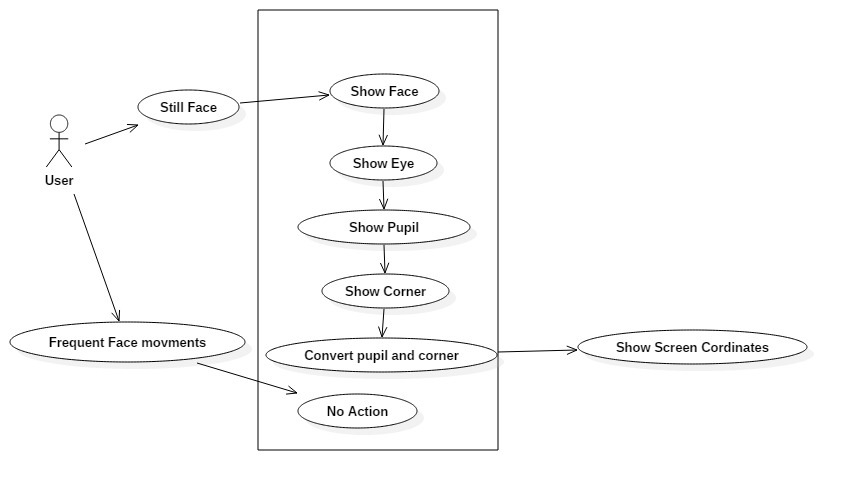
**Pupil Detection:**



**Corner Detection:**



**Screen Detection:**



**Heat map:**

