Final Report

PR Multimediale Systeme

SS 2016 – Gruppe [M27]

Document Control

Contributors

Bernhard Rieder	Developer	UNIVIE	Status Report
Bernhard Rieder	Developer	UNIVIE	Final Report
Michael Pichler	Developer	UNIVIE	Final Report

Revision History

Issue	Author	Date	Description
V0.1	Michael Pichler	12-Juni-2016	1st Objectives, Features
V0.2	Michael Pichler	13-Juni-2016	1st Technologies and Third-Party Software
V0.3	Michael Pichler	14-Juni-2016	1st Risk Analysis, Activites per Person, Status per Milestone, Installation Guidelines
V0.4	Michael Pichler	15-Juni-2016	1st Detailed Design (Interface, Screenshots), QuickStart Tutorial
V0.5	Michael Pichler	16-Juni-2016	1st Detailed Design (Architecture, Description), 2nd Progress Report
V0.6	Bernhard Rieder	17-Juni-2016	Updated Installation Guidelines
V0.7	Bernhard Rieder	17-Juni-2016	2nd Version of Objectives
V0.8	Bernhard Rieder	17-Juni-2016	2nd Version of Features
V1.0	Bernhard Rieder	17-Juni-2016	2nd Version of System Architecture, … & Final Version of Final Report

Table of Contents

1 Objectives

Das Ziel dieses Projektes ist es, einen funktionierenden GazeTracker zu implementieren und die Funktionen in einem Anwendungs-Szenario anzuwenden. Bestehende Systeme sind recht aufwendig und für die normale UserIn meist kaum leistbar, liefern aber dafür auch gute Resultate. Deswegen sollte der GazeTracker für dieses Projekt einfach und billig sein, aber dennoch eine akzeptable Blickerkennung gewährleisten. Um das Tool jeder NutzerIn zugänglich zu machen, wird auf die Hardware zurückgegriffen, die ein Großteil der UserInnen besitzt. Die Rede ist hierbei von einer gewöhnlichen Webcam, die heutzutage in fast jedem Notebook verbaut ist.

Die Anwendungsmöglichkeiten des GazeTrackers sind Vielfältig. Angefangen von der Einsatzmöglichkeit als kontaktlose Steuerung eines Systems mithilfe der Augen (z.B.: Steuerung eines Videoplayers, Objekte bewegen/auswählen, etc.), bis hin zur Analyse von Nutzungsverhalten.

Das Anwendungsgebiet, welches wir mit dem GazerTracker zeigen möchten, ist die Analyse des Nutzungsverhaltens einer UserIn hinsichtlich der Blickrichtung. Das Ziel ist es das Blickverhalten in einem Onlineshop wie zum Beispiel Amazon zu analysieren, um herauszufinden, welche Produkte wie lange von einer UserIn betrachtet werden und möglicherweise Interesse an einem Produkt hegen. Dabei soll die Zielgruppe eingeschränkt werden auf jene Personen, die keinen speziellen Kaufwunsch haben. Möglichkeiten der Analyseauswertung können, wie bereits beschrieben, sein, dass die Interessen an verschiedenen Produkten der NutzerIn ausgewertet werden können. Damit könnte bestimmt werden, was sich eine NutzerIn am ehesten kaufen würde oder benötigt.

Funktionale Anforderungen:

- Eye-Tracking
- Blickpunkt am Bildschirm bestimmen
- Kalibrieren für das verwendete System
- Daten-Aufzeichnung der NutzerIn zu jedem Zeitpunkt der aktiven Nutzung
    - Frame zu diesem Zeitpunkt
    - Blickposition zu diesem Zeitpunkt im Bezug zum dazugehörigen Frame

Nichtfunktionale Anforderungen:

- Benutzerfreundlichkeit
- Performanz
- Wiederverwendbarkeit

2 Features

Der GazeTracker soll für jeden PC mit einer Webcam und ausreichender Leistung (vor allem Grafikkarten-Leistung) ausführbar sein. Damit dieser Ordnungsgemäß funktioniert, ist es nicht nur wichtig, dass die Augen richtig erkannt werden, sondern auch die Erkennung der Blickposition muss zuverlässig funktionieren. Daher beinhaltet die Applikation ein Kalibrierungssystem, welches zu Beginn durchgeführt werden muss. Dabei wird die UserIn aufgefordert, ihre Iris zu markieren, eine Template Matching Methode auszuwählen und in bestimmte Positionen des Bildschirms zu blicken. Damit werden alle Komponenten des GazeTrackers für den verwendeten Bildschirm und der Umgebung kalibriert.

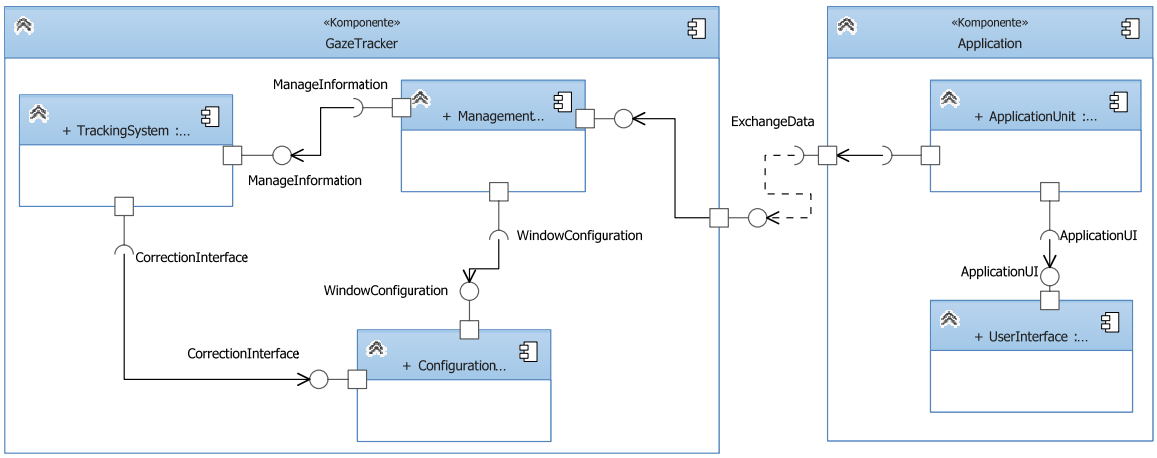
Die Interaktionsmöglichkeiten für die UserIn beschränken sich ausschließlich und hauptsächlich mit dem Kalibrierungsvorgang des eigentlichen Gazetrackers und der Verwendung der Analyse Applikation. Außerdem gibt es noch die Möglichkeit der Datenaufzeichnung, welche durch die

UserIn de/aktiviert werden kann. Sobald die UserIn die Kalibration durchgeführt und den Tracker gestartet hat, hat diese noch die Möglichkeit, sich die Blickpunkte am Bildschirm anzuzeigen oder auszublenden. Falls die Datenaufzeichnung aktiviert wurde, wird mit einer bestimmten Framerate der aktuelle Stand des Desktops abgespeichert. Zudem speichert der GazeTracker zu jedem Zeitpunkt die Position ab, wohin die UserIn gestarrt hat. Diese Informationen können nach Beendigung, mit der Analyse Applkation „GazeTracker Analytics“ zu einem Video zusammengefasst werden, indem die Veränderung der Blickpunkte begutachtet werden kann.

3 System Architecture

3.1 Design Overview

Dieser Abschnitt des Berichtes soll zeigen, welche Software bei der Realisierung verwendet wurde und wie darunterliegende Strukturen aussehen. Diese sind in verschiedenen Abstraktionsebenen gegliedert, angefangen vom Komponentendiagramm bis hin zu dem aktuellen User-Interface. Das System gliedert sich in 2 Komponenten. Die Erste wird als GazeTracker bezeichnet und bietet die beschriebene Hauptfunktionalität. Die zweite Komponente ist die Analyse Applikation, oder auch GazeTracker Analytics bezeichnet.



3.2 Detailed Design

3.2.1 Component GazeTracker

3.2.1.1 Description

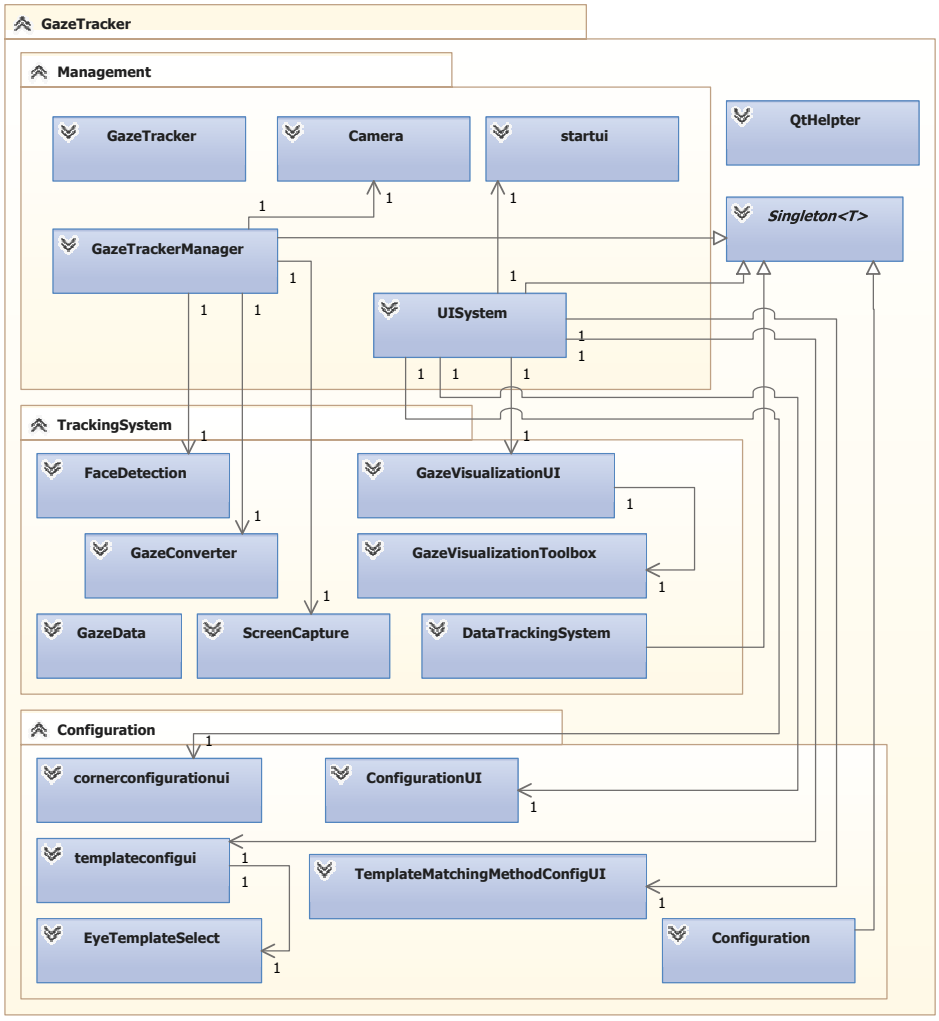
Diese Komponente bietet die Hauptfunktionalitäten des Systems an. Damit wird die Kamera angesprochen, mithilfe sogenannter „Cascade Classifier“ das Gesicht sowie Augen erkannt und für die weitere Verarbeitung aufbereitet. Mithilfe eines User Interfaces, welches in Kapitel 3.2.1.4 näher beschrieben wird, wird der UserIn der Bereich der Augen vorgeführt. Darin wird jene dazu aufgefordert den Bereich der Iris zu markieren, welcher im Folgenden als eigenes Bild abgespeichert wird. Diese Bilder werden im Weiteren für eine Template Matching Methode benötigt, die zur Ermittlung der Mittelpunkte der Iris verwendet wird. Sobald diese Mittelpunkte ermittelt wurden, muss die UserIn in die Eckpunkte ihres Hauptbildschirmes - mit der Kamera - blicken, um einen Referenzrahmen der Irispositionen in Relation zum Bildschirm zu schaffen. Der geschaffene Referenzrahmen wird schlussendlich für eine lineare Interpolation verwendet, mit der die Blickpunkte am Bildschirm ermittelt werden. Diese Blickpunkte können nach Wunsch als eigener Layer am Bildschirm angezeigt werden.

3.2.1.2 Documentation

- **GazeTracker**: Ist der Einstiegspunkt der Applikation.
- **Camera**: Ist die Schnittstelle zur Kamera und liefert die einzelnen Frames.

- **UISystem**: Ist eine Verwalterklasse für fast alle User Interfaces wie StartUI, ConfigurationUI, TemplateConfigUI, TemplateMatchingMethodConfigUI, GazeVisualizationUI und CornerConfigurationUI.
- **GazeTrackerManager**: Stellt das Gehirn des GazeTrackers dar und steuert alle einzelnen Klassen um im Endeffekt den Blickpunkt am Bildschirm zu ermitteln.
- **FaceDetection**: Ermittelt mithilfe Cascade Classifier die Gesichter und Augen. Liefert außerdem den Mittelpunkt der Iris.
- **GazeConverter**: Interpoliert die Iris Positonen zu Bildschirmpunkte.
- **GazeData**: Stellt die gespeicherten Daten dar, welche für den GazeTracker Analytics verwendet werden.
- **ScreenCapture**: Speichert je nach eingestellter FPS, den aktuellen Desktop als Bild am Laufwerk ab.
- **DataTrackingSystem**: Ist für die Speicherung der Daten als XML Datei verantwortlich.
- **Configuration**: Reine Speicherklasse für Applikationsrelevante Daten.

3.2.1.3 Architecture



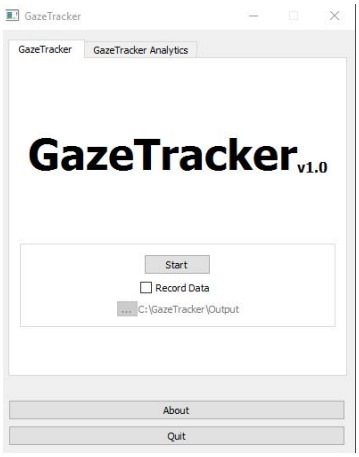
3.2.1.4 Interface

Das User Interface besteht aus einer StartUI, in der der GazeTracker gestartet werden kann. Außerdem kann hier eingestellt werden, ob die Daten gespeichert werden und wo diese gespeichert werden. Sobald gestartet wurde, wird man in die ConfigurationUI geführt, in der man Rückmeldung bekommt, ob die Kamera aktiv ist und ob ein Gesicht erkannt wurde. Sobald die ersten 2 Punkte fertiggestellt und ihre Farbe auf grün gewechselt haben, wird der Button für die

Augen Template Konfiguration freigeschalten. Hiermit wird ein Fenster geöffnet indem ein Rechteck über ein Bild des Auges gezeichnet werden kann, welches als Region Of Interest (ROI) dient. Wird dieses Fenster geschlossen, werden die ROI als Bilder abgespeichert und für die Template Matching Methode verwendet. Danach wird der Button für die Template Matching Methoden Konfiguration freigeschalten, in welchem die UserIn 1 aus 5 verschiedenen Methoden auswählen kann, welche am besten für sie funktioniert. Als Letztes wird der Button für die Ecken Konfiguration freigeschalten, indem die UserIn in jede Ecke blicken muss und ihren Augenkontakt mittels Buttons festsetzen muss. Als letzte UI erscheint ein durchsichtiges Fenster mit einer Toolbox, in der der Blickpunkt dargestellt wird. Mithilfe der Toolbox kann dies deaktiviert werden, die Applikation geschlossen und/oder die Konfiguration wiederholt werden.

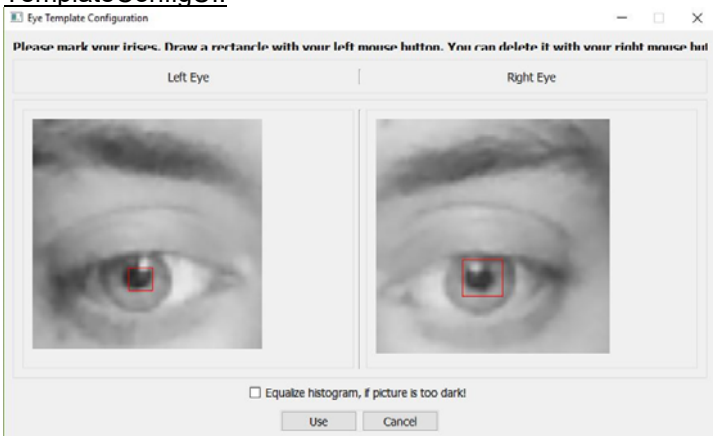
3.2.1.5 Screenshots

StartUI:

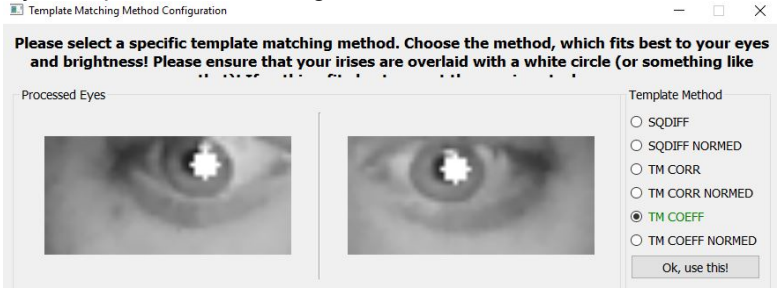


ConfigurationUI:

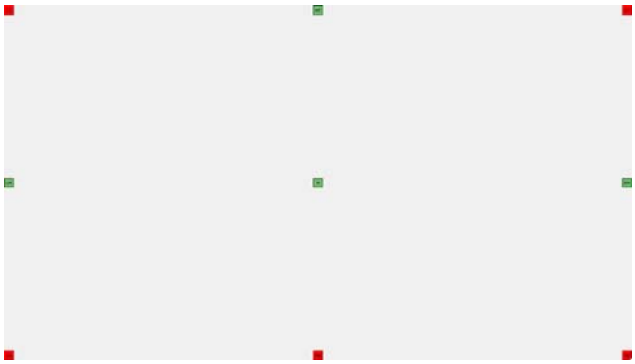
TemplateConfigUI:



EyeTemplateMethodConfigUI:



CornerconfigurationUI:



3.2.1.6 Objects and Actions

Mit Hilfe dieses Tools können Blickverhalten von verschiedenen NutzerInnen aufgezeichnet werden in Form von Frames und XML Daten. Ausgelöst wird diese Aktion nach dem Erkennen des Gesichtes und dessen notwendigen Informationen (Pupillen) und nach dem erfolgreichen konfigurieren des Systems.

3.2.1.7 Noteworthy

Die Genauigkeit des Systems muss mit Meilenstein 4 noch verbessert werden.

3.2.2 Component Application

3.2.2.1 Description

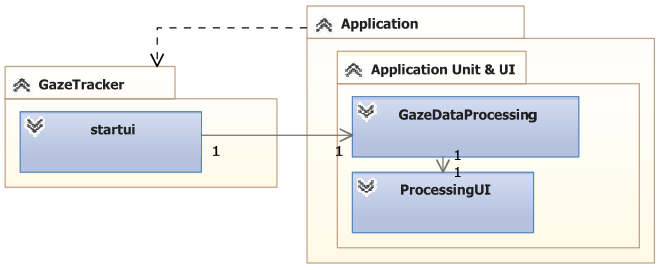
Die Applikation oder im Weiteren auch GazeTracker Analytics genannt, bietet eine Möglichkeiten die aufgezeichneten Daten als Video zusammenzufassen. Dafür werden die gespeicherten Frames und die mit Blickdaten gefüllte XML Datei verwendet um ein Video zu erstellen, indem die Blickpunkte visualisiert werden. Die Blickpunkte werden als grüne Kreise dargestellt, welche 1 Sekunde lang sichtbar sind. In dieser 1 Sekunde blenden sich diese Punkte aus, um aktuellere Punkte nicht zu überdecken.

3.2.2.2 Documentation

GazeTracker Analytics selbst, ist ein Teil des GazeTrackers und verwendet Teile des GazeTrackers.

- **StartUI**: Hier gibt es einen neben dem GazeTracker einen zweiten Reiter für die GazeTracker Analytics Applikation. Hier werden Auswahlmöglichkeiten für die XML und Frame Daten, sowie Output Ordner angeboten. Sobald diese Daten zur Verfügung gestellt wurden, wird die Analyse gestartet.
- **GazeDataProcessing**: Hier werden die XML Daten eingelesen und mit den einzelnen Frames zu einem Video zusammengeführt.
- **ProcessingUI**: Ist nur dazu da, um der UserIn Feedback zu geben, wie weit die Verarbeitung abgeschlossen ist. Dies ist nur ein einfacher Fortschrittsbalken und wird daher nicht gezeigt.

3.2.2.3 Architecture

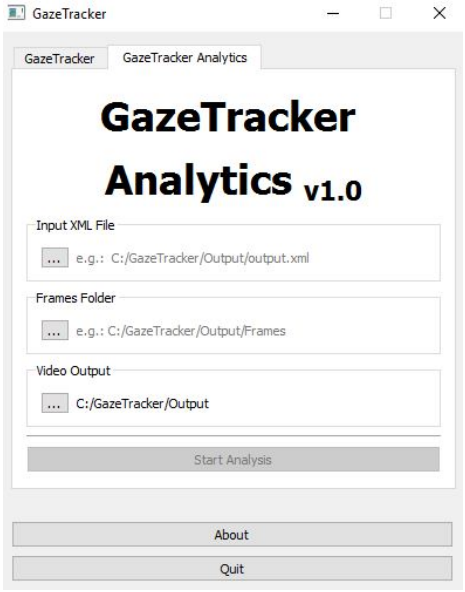


3.2.2.4 Interface

Das Programm startet mit der StartUI, indem die UserIn alle Daten für eine Verabeitung zur Verfügung stellen muss. Hierbei müssen die Pfade zu den XML und Frame Daten aus dem GazeTracker angegeben werden. Dazu muss jedoch im GazeTracker die Aufnahme der Daten aktiviert worden sein, da ansonsten keine Daten am Laufwerk gespeichert werden. Außerdem kann die UserIn den Ausgabe Ordner einstellen. Sobald alle Daten zur Verfügung gestellt wurden, kann die Analyse gestartet werden. Sobald sie gestartet ist, erscheint ein Fortschrittsbalken, in dem der Verarbeitungsstand gezeigt wird. Wenn 100% erreicht wurden, erscheint ein Info Dialog ob alles geklappt hat.

3.2.2.5 Screenshots

StartUI:



3.2.2.6 Objects and Actions

Ausgabe der Analyse ist ein Video im XVID Format, in dem die Blickpunkte zu den verschiedenen Zeiten weiter analysiert werden können.

3.3 Technologies and Third-Party Software

Für das Projekt GazeTracker mussten verschiedene APIs/Programme verwendet werden, damit die einzelnen Frames, welche die Webcam liefert, in Echtzeit verarbeitet werden konnten. Dafür gibt es eine Softwarebibliothek die genau für solche Probleme spezialisiert ist, nämlich die open Source Bibliothek für Computer Vision namens „OpenCV“. Ein großer Vorteil von OpenCV ist, dass diese Bibliothek für viele verschiedene Sprachen verwendet werden kann, welche wären:

- C, C++, Java, Python und viele mehr.

Für dieses Projekt wurde die OpenCV Version 3.1 mit der Sprache C++ gewählt. Der Hauptgrund für diese Entscheidung war, dass C++ eine Sprache ist, die direkt auf der Hardware läuft und eine hohe Performanz besitzt. Außerdem wurde OpenCV für C++ optimiert und erleichtert das Ansprechen der einzelnen Komponenten.

Eine weitere Bibliothek, die für dieses Projekt verwendet wurde, ist die für nicht-kommerzielle Zwecke frei verfügbar und verwendbare C++ GUI API „Qt“. Mithilfe Qt ist es möglich, schnell und einfach ein GUI Oberfläche zu konstruieren, mit den man die Kalibration und andere Interfaces den User bereitstellen kann. Natürlich hätte auch die WinForms API verwendet werden können, jedoch arbeitet OpenCV bereits mit Qt und daher bot sich die Zusammenarbeit mit Qt und OpenCV an.

Weiteres wurde für die Erstellung des Analyse Videos der XVID Codec verwendet. Wobei hier auch jeder andere Codec verwendet werden kann. Dieser wurde aber einfach aufgrund der niedrigen Speicherkapazität in Videos verwendet.

4 User Documentation

4.1 Installation Guidelines

Der GazeTracker wurde in Visual Studio 2015 und Windows 10 64 Bit entwickelt und getestet. Es wird daher empfohlen, diese Versionen zu verwenden.
In GazeTracker Analytics wird außerdem ein Video im XVID Codec erstellt, dessen Installation am Computer durchaus eine Vorraussetzung der problemlosen Anwendung ist.

Installationsvorbereitungen - Downloads:

- OpenCV für Windows - Version 3.1 (http://opencv.org/downloads.html)
- CMake 3.5.1 (https://cmake.org/download/)
- Python 3.5 im Paket "Anaconda für Windows"(https://www.continuum.io/downloads)
- Qt 5.6.0 für Windows 64 Bit und Visual Studio 2015 (https://www.qt.io/download-open-source/#section-2)
- Qt5Package - Visual Studio Extension (zu finden in Visual Studio 2015 unter "Extras/Extensions und Updates..")

Installationsanleitung:

1. OpenCV 3.1 in ein beliebiges Verzeichnis extrahieren - zB: „C:\OpenCV“. Darin befinden sich die Ordner „sources“ und „build“. In diesem Verzeichnis einen neuen Ordner „mybuild“ o.ä. erstellen, der später für den eigenen Build verwendet wird.
2. Qt 5.6.0 mit dem Installer installieren. Hierbei einfach die Standard Einstellungen der Installation verwenden. (Module, welche mit deprecated markiert sind, können natürlich weggelassen werden)
3. Umgebungsvariable „QTDIR“ für Qt erstellen, welche auf den Ordner „msvc2015_64“ im Installationsverzeichnis von Qt zeigt - zB: „C:\Qt\Qt5.6.0\5.6\msvc2015_64“.
4. Umgebungsvariable „%QTDIR%\bin“ in die PATH Systemvariable hinzufügen.
5. CMake mit Standard Einstellungen installieren und zur PATH Systemvariable hinzufügen lassen.
6. PC neu starten damit die Variablen aktiv werden.
7. CMake starten.
8. In „Where is the source code:“ das „source“ Verzeichnis von OpenCV setzen - zB: „C:/OpenCV /sources“.
9. In „Where to build the binaries:“ das „mybuild“ Verzeichnis von OpenCV setzen - zB: „C:/OpenCV /mybuild“.
10. Auf „Configure“ klicken. Hier erscheint ein Dialog, darin wählt man „Visual Studio 14 2015 Win64“ aus, damit OpenCV mit 64 Bit erstellt wird. Außerdem wählt man hier „Use default native compilers“ aus und bestätigt den Dialog.
11. Es erscheint nun eine Liste mit Einstellungsoptionen, wobei die meisten oder alle rot hinterlegt sind. Hier werden die Optionen „BUILD_DOCS“, „BUILD_opencv_python2“, „BUILD_opencv_python3“ und „WITH_VTK“ abgewählt. Um nun die Unterstützung für Qt im OpenCV Build zu haben, muss „WITH_QT“ angewählt werden.
12. Nun muss einige Male auf „Configure“ gedrückt werden (ca. 3x), bis alle rot hinterlegten Einträge verschwunden sind. Hier kann es sein, dass die PATH Variable zu Qt nicht gefunden wird, was dann als rot hinterlegter Eintrag erscheint. Diese muss wie in Schritt 3 und 4 beschrieben, hinzugefügt werden.
13. Sobald alles weiß hinterlegt ist, kann auf „Generate“ gedrückt werden, danach wird der Build erstellt.
14. CMake kann nun beendet werden.

15. Nun wechselt man in das Verzeichnis des Builds (C:\OpenCV\mybuild) und öffnet „OpenCV.sln“. Hier sollte der Modus auf Debug und x64 eingestellt sein, da vorhin für 64 Bit ein Build erstellt wurde.
16. Darin das Projekt „ALL_BUILD“ suchen, mit der rechte Maustaste darauf klicken und Erstellen auswählen. (Damit werden die *.lib und *.dll Daten für das Debugging erstellt) Hierbei sollten bei dem Erstellen keine Fehler auftreten. Danach wird der Modus auf Release geändert und für den Release Modus das Projekt „ALL_BUILD“ nochmals erstellt. Hierbei sollten wieder keine Fehler auftreten.
17. Hat alles geklappt, klickt man mit der rechten Maustaste auf „INSTALL“ und Erstellen. Dies ist unter „CMakeTargets“ zu finden. Der Schritt wird wie vorhin für Debug und Release extra ausgeführt. Mithilfe diesen Schrittes, werden alle *.lib und *.dll Daten in einen eigenen Ordner verschoben, welcher unter „..\mybuild\install\x64\vc14\..“ zu finden ist.
18. Den Ordner „install“ kopiert man nun in das OpenCV Verzeichnis (C:\OpenCV).
19. Um OpenCV einheitlich verwenden zu können wird eine Umgebungsvariable „OPENCV_DIR“ angelegt, welche auf den Ordner „install\x64\vc14“ aus dem OpenCV Verzeichnis zeigt.
20. Des Weiteren muss in der PATH Systemvariable ein Eintrag mit „%OPENCV_DIR%\bin“ ergänzt werden.
21. PC neu starten damit die Variablen aktiv werden.
22. OPTIONAL: In Visual Studio 2015 muss eventuell noch die Qt Version mit dem zugehörigen Pfad eingetragen werden, damit die Qt Funktionen verfügbar sind. Dazu öffnet man in Visual Studio 2015 den Menu Punkt „QT5/QT Options“. Falls hier schon die aktuellste Version eingetragen ist, muss man nichts mehr machen. Ansonsten klickt man auf „Add“ und schreibt den Versionsnamen und fügt den Pfad für den „mcsv2015_64“ Ordner ein - z.B: "C:\Qt\Qt5.6.0\5.6\msvc2015_64".

Nach dieser Installationsanleitung sollte sich das Projekt in Visual Studio 2015 ausführen lassen! Sollte dies nicht der Fall sein, sollte die Anleitung wiederholt werden und auf die genauen Versionen und Anweisungen geachtet werden.

4.2 Quick Start Tutorial

Dieser Abschnitt beschäftigt sich damit, wie dieses Tool zu bedienen ist.

Als erstes muss die UserIn die richtige Umgebung für den GazeTracker schaffen. Der GazeTracker benötigt ein Gesicht, welches von vorne gut beleuchtet ist, damit die Iris Positionen genau erkannt werden können. Außerdem darf sich die UserIn während der kompletten Anwendung nicht bewegen, da der GazeTracker keine Bewegungskompensation besitzt.

Am Beginn wird ein Fenster gezeigt (siehe StartUI) mit zwei Tabs, wo diverse Einstellungen getätigt werden können. Der Linke Tab ist der eigentliche GazeTracker und der rechte Tab ist die Analyse Applikation.

4.2.1 GazeTracker

Für den GazeTracker können Einstellungen getätigt werden, ob Daten gespeichert werden und wo diese gespeichert werden. Sobald auf „Start“ geklickt wird, öffnet sich die Configuration UI indem der UserIn Rückmeldung gegeben wird, ob die Webcam erkannt wurde und ob ein Gesicht erkannt wurde. Sobald dies geschehen ist, schaltet sich der Button für die Augen Template Auswahl frei, in der die UserIn ein Rechteck über ihre Pupillen ziehen muss (siehe TemplateConfigUI). Wurde dies gemacht, kann auf „Use“ geklickt werden, um die gezeichneten Bereiche zu verwenden. Danach muss eine Template Matching Methode ausgewählt werden, die zu der UserIn passt. Hierbei muss die UserIn auf der rechten Seite eine Checkbox auswählen, welche am besten auf der Iris einen Kreis zeichnet (siehe TemplateMatchingMethodConfigUI). Ist dies ausgewählt muss am Schluss nur noch die Ecken Konfiguration ausgeführt werden. Dazu muss einfach in jede Ecke geblickt werden und dann auf die jeweligen Buttons geklickt werden. Wurde eine Ecke akzeptiert, wird der Button in der Farbe Grün dargestellt, ansonsten in Rot. Jetzt kann die richtige Applikation gestartet werden. Hier gibt es ein transparentes Fenster und eine Toolbox. Im transparenten Fenster werden die Blickpunkte als grüne Ellipse dargestellt. Diese können in der Toolbox abgeschalten werden.

Außerdem kann in der Toolbox die Konfiguration wiederholt werden und die Applikation geschlossen werden.

Das linke Bild zeigt an wie es aussieht, wenn der GazeTracker gestartet wird. Von diesem Fenster aus kann mithilfe des Start-Buttons die Konfiguration gemacht werden die immer notwendig ist vor der Nutzung. Beim linken Bild können die einzelnen Pfade angegeben werden, woe die Daten gespeichert werden sollen (wo das zu analysierende Video gespeichert werden soll, etc.).

Wie vorhin erwähnt, muss der GazeTracker vor der Nutzung konfiguriert werden, dazu gibt es verschiedene Schritte, die gemacht werden müssen.

4.2.2 GazeTracker Analytics

Der zweite Tab der StartUI beinhaltet die Analyse Applikation. Hier muss die UserIn die XML und Frame Daten, welche zuvor im GazeTracker aufgenommen wurden, zur Verfügung stellen. Außerdem kann der Ausgabe Ordner eingestellt werden. Sobald auf Start geklickt wird, startet die Analyse und erstellt im Ausgabe Ordner ein Video, welches für die weitere Analyse verwendet werden kann. Darin erscheinen die Blickpunkte jeweils 1 Sekunde.

5 Progress Report

5.1 Summary

In diesem Abschnitt wird auf den Fortschritt des Projektes näher eingegangen und zwar in der Einteilung der Meilensteine:

Meilenstein 1:

Dieser Abschnitt des Projektes war am Aufwändigsten, denn hier musste die Entscheidung getroffen werden, welche externe Software verwenden soll und das ganze Projekt auszusetzen. Es war klar, dass für die Erkennung der Augen/Pupillen mit OpenCV verwendet wird, da diese Bibliothek vorgefertigte Classifier für die Gesichts- und Augenerkennung beinhaltet. Probleme gab es bei der Gesichtserkennung deshalb, weil im nachfolgendem Schritt die Augen nicht zuverlässig genug erkannt wurden. Deshalb wurde entschieden, dass man das Gesicht in zwei Hälften unterteilt und dann jeweils einen speziellen Classifier für die Augen verwendet (einen für das Rechte und einen für das Linke).

Auch die Pupillenerkennung im Eye-Frame hatte zu Beginn große Probleme bereitet. In Meilenstein 1 wurden vorerst verschiedene Filter verwendet, um den Mittelpunkt des Auges zu detektieren. Die Methode mit verschiedenen Filtern wurde jedoch verworfen, da OpenCV auch die Methode des Template Matching zur Verfügung stellt, welche bessere Ergebnisse in der Pupillenerkennung brachte.

Meilenstein 2:

Im Meilenstein 2 ging es darum, den Punkt, auf den die UserIn gerade schaut, auf dem Bildschirm zu berechnen. Dazu musste eine GUI entwickelt werden, wo zuerst diverse Einstellungen getroffen werden können. Nach der Konfiguration wird der Punkt berechnet. Dabei gibt es Probleme, wie genau die Berechnung mit dem Blickpunkt übereinstimmt. Es gibt einen Faktor, der für uns nicht beeinflussbar ist, nämlich wie hoch die Auflösung der Webcam ist. Je höher diese ist, desto genauer stimmt der Blickpunkt. Jedoch bringt eine höhere Auflösung eine höhere Rechenleistung.

Meilenstein 3:

Dieser Meilenstein war einer der Leichtesten, da hier nur die XML und Frame Daten zusammengefügt wurden. Die Analyse des Videos ist ein manueller Teil, der mit keinem Programm durchgeführt werden kann.

Meilenstein 4:

In diesem Meilenstein geht es darum, bis zur Präsentation des GazeTrackers kleinere Verbesserungen durchzuführen. Zum Beispiel kann muss noch an der Genauigkeit der GazeTrackers gearbeitet werden, da diese zurzeit nicht immer sehr verwendbar ist. Außerdem könnte noch eine Bewegungskompensation implementiert werden.

5.2 Status per Milestone

Meilenstein 1:

Der Meilenstein 1 wurde zeitgerecht begonnen und ist beinahe zu 100% beendet. Der Zugriff der Kamera funktioniert einwandfrei, was als Voraussetzung des ganzen Projektes ist. Die Gesichts- und Augenerkennung wurde implementiert und funktioniert schon relativ gut. Die Pupillen werden auch schon erkannt und die Position ermittelt. Hier gibt es noch Verbesserungspotenzial in Hinsicht auf verschiedene Lichtverhältnisse und Position (Da diese noch zu stark hin und her springt).

Das einzige, was noch nicht funktioniert ist die Erkennung, in welche Richtung die Person gerade blickt. Dieser Punkt wird somit in den Meilenstein 2 verschoben, da es hier hauptsächlich darum geht, den GazeTracker zu kalibrieren und den Auftreffpunkt des Blickes zu ermitteln.

Meilenstein 2:

Auch dieser Meilenstein wurde zeitgerecht begonnen und vollständig fertiggestellt. Auch der Teil, welche vom Meilenstein 1 zu Meilenstein 2 verschoben wurde, ist fertiggestellt worden.

Meilenstein 3:

Der Showcase wurde ebenfalls zeitgereicht angefangen und fertiggestellt, da dieser keinen großen Aufwand hatte.

Meilenstein 4:

Dieser Meilenstein ist bis zur Endpräsentation aktiv und beinhaltet kleinere Verbesserung vom GazeTracker, um diesen vorführen zu können

5.3 Risk Analysis

Da dieses Projekt als beendet eingestuft wird, können in der Entwicklung keine Risiken mehr vorkommen. Wie im Status-Report erwähnt, wurde die Methode für die Augenerkennung umgestellt, um Performanz und Genauigkeit bei der Erkennung zu steigern. In der finalen Version wird die Template-Methode verwendet.

Ein anderes Risiko, welches von der Seite der Programmierer nur beschränkt beseitigt werden kann sind die Lichtverhältnisse in der Anwendungsumgebung. Dafür muss der Benutzer unseres Tools sorgen damit der GazeTracker ordnungsgemäß funktioniert.

Ein Punkt, was hier noch zu erwähnen ist, dass dieses Tool nur ausgelegt ist für eine Person. Sollten mehrere Personen erkannt werden, führt diese zu einem Fehler bei der Erkennung der richtigen Blickposition. Daher sollte immer nur eine Person im Blickfeld der Kamera sein.

5.4 Activities per Person

Es wurde versucht, die Aufgaben so gut wie möglich untereinander aufzuteilen. Die Kommunikation zwischen den Personen in der Gruppe hat zumeist über Facebook und Skype stattgefunden, um die nächsten Schritte zu besprächen oder anstehende Aufgaben einzuteilen. Am Ende des Meilensteins gab es ein kurzes Zusammentreffen, um die weitere Schritte zu besprechen.

5.4.1 Report on working hours per person

Anforderungsanalyse (100% = 20 PS) PS = Personen Stunden

- Bernhard: 50% (10 PS)
- Michael: 50% (10 PS)

Meilenstein 1 (100% = 100 PS)

- Aufsetzen des Projektes und Zugriff auf die Kamera 20%
    - Bernhard: 10% (10 PS)
    - Michael: 10% (10 PS)
- Face and Eye Detection: 35%
    - Bernhard: 17,5% (17,5 PS)
    - Michael: 17,5% (17,5 PS)
- Pupil Detection: 45%
    - Bernhard: 22,5% (22,5 PS)
    - Michael: 22,5% (22,5 PS)

Statusreport (100% = 6 PS)

- Bernhard: 50% (3 PS)
- Michael: 50% (3 PS)

Meilenstein 2 (100% = 80 PS)

- Bernhard: 60% (48 PS)
- Michael: 40% (32 PS)

Meilenstein 3 (100% = 50 PS)

- Bernhard: 75% (37,5 PS)
- Michael: 25% (12,5 PS)

Meilenstein 4 (100% = 50 PS) -> Fortlaufend

- Bernhard: 50% (25 PS)
- Michael: 50% (25 PS)

Final Report (100% = 6 PS)

- Bernhard: 50% (3 PS)
- Michael: 50% (3 PS)