Blink Detector

Seminararbeit

vorgelegt von

Jan-Nicolas Weider, Felix Tischler und Mattis Dietrich Dietrich

angefertigt am

Lehrstuhl für Digitale Bildverarbeitung
Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena

Seminar: Seminar Rechnersehen
Betreuer: Niklas Penzel
Datum: 25. September 2023

Inhaltsverzeichnis

1 Einführung **2**

2 Theorie **2**

2.1 Gesichtserkennung 2
2.2 Blinzeldetektion 2
2.3 Features 2
2.4 Klassifizierung 2
2.5 App Entwicklung 2
2.5.1 Technologieauswahl 2
2.5.2 Entwicklungsprozess 3
2.5.3 Deployment-Strategie 4

3 Datensatz **5**

4 Evaluation **5**

4.1 Das Interface/Die App 5
4.1.1 Realisierung 5

5 Lessons Learned **6**

5.1 Herausforderungen in der App-Entwicklung 6
5.2 Deployment-Herausforderungen 8

6 Zusammenfassung **9**

Literatur **10**

Abbildungsverzeichnis **11**

1 Einführung

2 Theorie

2.1 Gesichtserkennung

2.2 Blinzeldetektion

2.3 Features

2.4 Klassifizierung

2.5 App Entwicklung

Unser Projekt hatte zum Ziel, eine Anwendung zu entwickeln, die die Müdigkeitserkennung durchführt. Diese App sollte auf verschiedene Plattformen und Geräte zugänglich sein und es den Benutzern ermöglichen, ihre Müdigkeit in Echtzeit zu überwachen. Die App sollte einfach zu bedienen, effizient und zuverlässig sein.

2.5.1 Technologieauswahl

Zu Beginn der Technologieauswahl stand für uns schon fest (wie sich später noch als unüberlegt herausstellen sollte): Python ist das Mittel der Wahl für unsere Idee und Umsetzung aller benötigten Algorithmen. Wir machten uns also von Anfang an auf die Suche nach geeigneten Symbiosen die uns die Entwicklung einer App mit Hilfe von Python ermöglichen.

Als folge dessen haben wir uns bei der Auswahl der weiteren Technologien für unser Projekt intensiv mit den verschiedenen Optionen auseinandergesetzt. Eine entscheidende Überlegung war die Notwendigkeit der Offline-Funktionalität, da unsere App in Umgebungen zum Einsatz kommen sollte, in denen eine dauerhafte Internetverbindung nicht immer gewährleistet ist. Aus diesem Grund haben wir uns zum Beispiel bewusst gegen die Realisierung einer API mit React und Flask entschieden.

An dieser Stelle sei erwähnt dass die Entscheidung gegen soeben erwähntes nicht so leichtfertig viel wie der vorherige Satz vermuten mag. Hinter der Entscheidung stand ein intensives Erproben und Evaluieren der Möglichkeiten mit React und Flask. Schlussendlich war uns klar dass man nur zu benannten Konklusion kommen kann.

Es musste eine andere Lösung her und nach genauerer Recherche (Erprobung und Evaluation) viel die Wahl auf Kivy. Die Entscheidung für das Kivy-Framework als Frontend-Framework war fast schon eine zwingende Konsequenz auf Grund von vielen Vorteilen. Kivy Apps können auf verschiedenen Systemen zum Einsatz kommen und bieten somit eine breite Einsatzmöglichkeit. Zusätzlich synergiert Kivy mit Python und wurde einst für diese Programmier Sprache entwickelt. Dies legte final den Grundstein für die Verbindung unseres Frontend mit unserer Logik die ausschließlich in Python liegt.

Die Wahl von Kivy als notwendige Konsequenz zur Wahl von Python ermöglichte es uns unser Projekt in den verschiedenen Fassetten zu starten und zu entwickeln. In den folgenden Abschnitten werden wir auf den Entwicklungsprozess, die Implementierung und die Herausforderungen bei der Bereitstellung der App näher eingehen.

2.5.2 Entwicklungsprozess

Zu Beginn des Entwicklungsprozess war zwar klar welche Technologien wir verwenden wollen, jedoch blieb die Frage der Implementierung noch aus. Es bildeten sich drei Teilbereiche welche diese Frage beantworten konnten:

- Entwicklung des App Grundgerüst
- Entwicklung des Layout der App
- Entwicklung der Integration von Logik und Layout

Ersteres definierte notwendige Schnittstellen und legte den Rahmen fest in dem wir agierten. Es wurde eine Hauptdatei erstellt (mainmediapipe.py) welche die Logik des Backend aus einer externen Datei und ebenso das Layout beinhaltete und verschmelzen ließ. Zweiteres hingegen Setzte alle notwendigen Design Entscheidungen fest in einer Datei (mainmediapipe.kv) hierfür verwendeten wird das von Kivy (für diese Zwecke) mit gelieferte Dateiformat '.kv'. Letzteres beanspruchte mit Abstand an meisten Zeit, as mussten Verlinkungen zwischen der Logik und dem Layout vorgenommen werden. Es wurden Mechanismen für die visuelle und akustische Alarmierung entwickelt. Alles Schnittstellen wurden sozusagen in Betrieb genommen und mussten untereinander kommunizieren können. Wir haben mittels der uns zur Verfügung stehenden Möglichkeiten all diese Hürden bezwingen können und ein funktionierendes Gesamtpaket entstehen lassen, als Produkt des Entwicklungsprozess.

Der Stand der App war zu diesem Zeitpunkt rein Desktop basiert. Im Lessons Learned-Teil werden wir näher darauf eingehen, warum wir uns gegen die Fortsetzung der Bereitstellung bis zur Erstellung einer APK-Datei entschieden haben. Dabei werden wir auf die Herausforderungen und Lernprozesse eingehen, die diese Entscheidung beeinflusst haben.

2.5.3 Deployment-Strategie

Das große Ziel war ein funktionierendes Stück Software zu haben die unseren Projekt Anforderungen gerecht wird. Darüber hinaus hatten wir noch einen weiteren selbst gewählten Anspruch: wir wollten eine App entwickeln, die wenn möglich so weit wie nur erreichbar verfügbar ist. Das heißt konkret das das Ziel war sich sukzessive von Softwarecode, zu Desktop App bis hin zur APK Datei zu verbessern. Dies war ein Vorhaben das über die Anforderungen des Modul hinaus ging.

Start der Deployment-Strategie war es folgende drei Faktoren zu betrachten:

- Zielplattformen und -geräte
- Deployment-Tools und -Methoden
- Herausforderungen bei der Bereitstellung

Zielplattformen und -geräte sollten zunächst Windows und Linux abbilden und in zweiter Instanz Android. Deployment-Tools und -Methoden waren und durch unsere Technologieauswahl bereits bindend vorgegeben. Herausforderungen bei der Bereitstellung ergaben sich dann zu häuft, denn die zunächst naiv eingeschätzte Möglichkeit eine App möglichst unkompliziert bereit zu stellen stellte sich als massive Fehleinschätzung heraus. Wir konnten relativ unkompliziert unseren Code als Desktop App anbieten der auf Windows und Linux funktionierte. Jedoch war der weg zur APK Datei wesentlich widerstandsfähiger. Es folgten diverse versuche einen Build Prozess anzustoßen der unseren Code zur benötigten Datei umbauen sollte. Jedoch hatten wir einen entscheidenden Fehler in der Technologieauswahl begangen: wir legten uns von Anfang an auf Python fest und limitierten uns damit ausschlaggebend in der Wahl der Build Möglichkeiten. Für eine nähere Betrachtung dieses Fehlers verweisen wir jedoch in unseren Lesson Learnd Teil auf den Eintrag 5.2.

3 Datensatz

4 Evaluation

4.1 Das Interface/Die App

4.1.1 Realisierung

Die App bietet eine Benutzeroberfläche mit verschiedenen Bildschirmen: Dies ist der

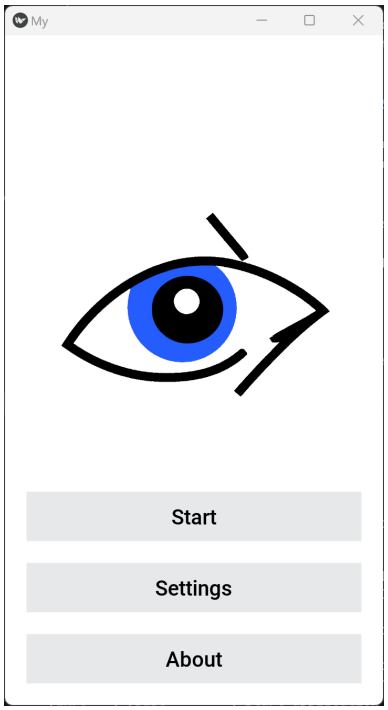

Abbildung 1: Hauptbildschirm (MainScreen)

Einstiegsbildschirm der App. Hier erhalten Benutzer eine kurze Einführung in die App und können die Müdigkeitserkennung starten oder auf die Einstellungen zugreifen. Hier findet die eigentliche Müdigkeitserkennung statt. Die Kamera zeigt eine Echtzeitansicht, und erkannte Gesichtsmerkmale wie die Augen werden markiert. Bei Erkennung von Müdigkeitsmerkmalen kann die App Warnungen anzeigen oder Alarmtöne abspielen. Benutzer können hier verschiedene Einstellungen für die Benutzeröberfläche anpassen. Dieser Bildschirm bietet detaillierte Informationen zur Verwendung der App und ihrer Funktionen.

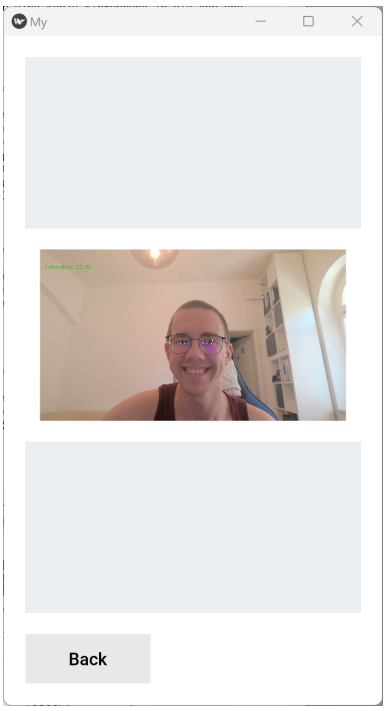

Abbildung 2: Erkennungsbildschirm (DetectionScreen)

5 Lessons Learned

5.1 Herausforderungen in der App-Entwicklung

Unsere Reise in der App-Entwicklung war von zahlreichen Herausforderungen und Lernprozessen geprägt, die unser Verständnis für die Entwicklung von plattformübergreifenden Anwendungen vertieft haben. Eines der wichtigsten Erkenntnisse, die wir gewonnen haben, war die Bedeutung einer umfassenden Betrachtung aller Faktoren, bevor wir uns auf eine bestimmte Technologie festlegen. In unserem Fall war es ein Fehler, sich frühzeitig auf Python als Programmiersprache festzulegen, ohne die Alternativen zu prüfen. Eine gründliche Analyse aller relevanten Faktoren, einschließlich der Eignung der Programmiersprache, hätte uns möglicherweise zu besseren Entscheidungen geführt.

Wir haben nach Abschluss des Projekt erkannt, dass Java und Kotlin als Programmiersprachen besser zu unseren Anforderungen gepasst hätten. Beide Sprachen bieten nicht nur eine reibungslose Integration der benötigten Bibliotheken wie OpenCV und Mediapipe, sondern sind auch in Bezug auf die App-Entwicklung insgesamt charmanter. Diese Erkenntnis hat uns gezeigt, wie wichtig es ist, die Flexibilität bei der Wahl der Technologie zu wahren und die Optionen offen zu halten.

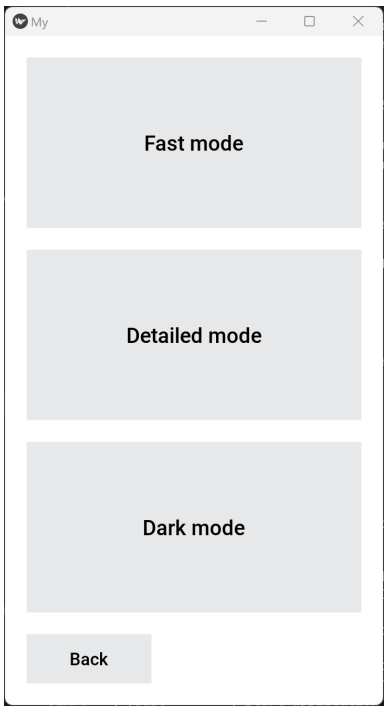

Abbildung 3: Einstellungsbildschirm (SettingsScreen)

Ein weiterer wesentlicher Lernpunkt war die Priorisierung von Design und Usability. In unserem Fall lag die Hauptaufmerksamkeit zu stark auf der reinen Integration der Müdigkeitserkennungslogik und dem Versuch, die App zu deployen. Dabei hätten wir mehr Ressourcen und Aufmerksamkeit auf das Design und die Benutzerfreundlichkeit der App und des Interfaces lenken können.

Darüber hinaus haben wir erkannt, dass die App-Entwicklung noch tiefer gehen kann, insbesondere im Hinblick auf die Nutzung der Hardware des Endgeräts. Wir könnten spezielle Techniken zur Verbesserung der Stabilität und Geschwindigkeit der App erforschen. Dies umfasst die effiziente Nutzung von Ressourcen wie Kamera und Prozessor, um eine reibungslose Erfahrung für die Benutzer sicherzustellen.

Insgesamt haben diese Herausforderungen und Lernprozesse unser Verständnis für die Entwicklung von Apps erweitert und uns dabei geholfen, fundierte Entscheidungen zu treffen. Diese Erkenntnisse werden uns in zukünftigen Projekten dabei helfen, die Qualität unserer Apps weiter zu verbessern und die Anforderungen hypothetischer Benutzer optimal zu erfüllen.

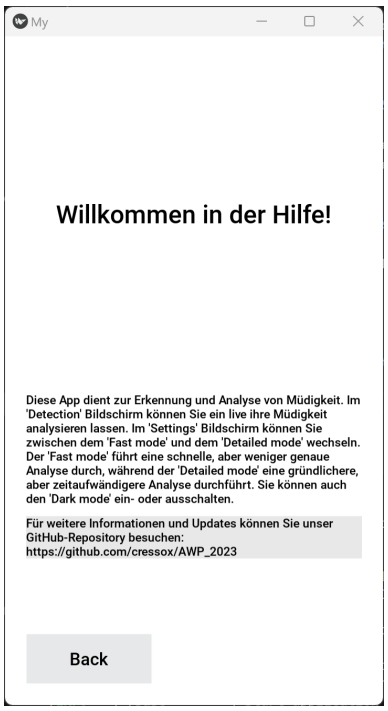

Abbildung 4: Hilfebildschirm (HelpScreen)

5.2 Deployment-Herausforderungen

Eine der herausforderndsten Phasen unseres Projekts war zweifellos der Versuch, die App als APK-Datei bereitzustellen. Diese Herausforderungen ergaben sich aus unserer anfänglichen Entscheidung, Python und das Kivy-Framework zu verwenden, was sich als eine limitierende Faktoren für das Deployment herausstellte.
Ein bedeutendes Problem war die Unausgereiftheit des Build-Prozesses mit Kivy im Vergleich zu etablierten Alternativen wie Android Studio in Verbindung mit Java oder Kotlin. Der Build-Prozess mit Kivy war anfälliger für Fehler und Schwierigkeiten. Je mehr Abhängigkeiten und Bibliotheken in unserer Logik verwendet wurden, desto geringer war die Wahrscheinlichkeit, dass der Build-Prozess ohne Fehler durchlief.
Wir haben schmerzlich erfahren, dass die Wahl von Kivy für unser Projekt die Möglichkeit, die App als APK-Datei bereitzustellen, stark eingeschränkt hat. Dies führte zu erheblichem Zeitaufwand und frustrierenden Versuchen, Lösungen zu finden. Wir unternahmen eine Vielzahl von Rettungsversuchen, darunter das Deployment über Docker und die Implementierung in einem Container. Wir gingen sogar so weit, die grundlegende Funktionsweise von Kivy beim Kompilieren und Builden tiefgehend zu analysieren.
Die Zeit, die in diese Bemühungen investiert wurde, hätte stattdessen genutzt werden können, um die Potenziale zu erforschen, die bereits in Kapitel 3 erwähnt wurden. Eine

verstärkte Fokussierung auf die Verbesserung des Designs, der Usability und der Effizienz der App sowie die tiefere Integration der Hardware des Endgeräts hätte unsere App insgesamt verbessern können.
Diese Deployment-Herausforderungen haben uns wertvolle Einsichten vermittelt, darunter die Notwendigkeit, den Build-Prozess frühzeitig zu evaluieren und eine gut durchdachte Wahl der Entwicklungsumgebung zu treffen. Sie haben uns auch gezeigt, wie wichtig es ist, die Abhängigkeiten und Anforderungen der ausgewählten Technologie sorgfältig zu berücksichtigen, um unerwartete Schwierigkeiten im späteren Entwicklungsprozess zu vermeiden.

6 Zusammenfassung

Literatur

Abbildungsverzeichnis

1 Hauptbildschirm (MainScreen) 5
2 Erkennungsbildschirm (DetectionScreen) 6
3 Einstellungsbildschirm (SettingsScreen) 7
4 Hilfebildschirm (HelpScreen) 8