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: 30. September 2023

Inhaltsverzeichnis

1 Einführung

2 Theorie

2.1 Gesichtserkennung

2.2 Blinzeldetektion

2.3 Features

2.4 Klassifizierung

2.5 App Entwicklung

Unser Projekt hat zum Ziel, eine Anwendung zu entwickeln, die die Müdigkeitserkennung durchführt. Diese App sollte auf verschiedenen Plattformen und Geräten 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 bereits fest, dass Python das Mittel der Wahl für unsere Idee und Umsetzung aller benötigten Algorithmen ist. Wir begaben uns 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 bewusst gegen die Realisierung einer API mit React und Flask entschieden. Hinter der Entscheidung stand ein intensives Erproben und Evaluieren der Möglichkeiten mit React und Flask. Schlussendlich wurde jedoch deutlich, dass eine andere Lösung gefunden werden muss.

Nach genauerer Recherche, Erprobung und Evaluation fiel die Wahl auf Kivy. Die Entscheidung für das Kivy-Framework zur Fronent-Entwicklung war eine nehzu 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 Programmiersprache entwickelt.

Dies legte final den Grundstein für die Verbindung unseres Frontends mit unserer Logik, die ausschließlich auf Python basiert. Die Wahl von Kivy ermöglichte es uns, unser Projekt in den verschiedenen Fassetten starten und entwickeln zu können.

2.5.2 Entwicklungsprozess

Zu Beginn des Entwicklungsprozess war zwar entschieden, 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-Grundgerüsts
- Entwicklung des Layouts 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 kombinierte. Zweiteres hingegen setzte alle notwendigen Entscheidungen bezüglich des Designs in einer Datei (mainmediapipe.kv) fest. Hierfür verwendeten wird das von Kivy mitgelieferte Dateiformat '.kv'. Letzteres beanspruchte am meisten Zeit, es mussten Verlinkungen zwischen der Logik und dem Layout vorgenommen werden. Es wurden Mechanismen für die visuelle und akustische Alarmierung entwickelt. Alles Schnittstellen wurden in Betrieb genommen und mussten untereinander kommunizieren können. Wir haben mittels der uns zur Verfügung stehenden Möglichkeiten all diese Hürden bezwingen können und als Produkt des Entwicklungsprozesses ein funktionierendes Gesamtpaket entstehen lassen.

Zu diesem Zeitpunkt war die App rein desktopbasiert. Im Abschnitt 5.2 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, eine funktionierende Software zu haben, die unseren Projektanforderungen gerecht wird. Darüber hinaus hatten wir noch einen weiteren selbstgewählten Anspruch, wir wollten eine App entwickeln, die sich über eine maximale Verfügbarkeit auszeichnet. Das heißt konkret, das Ziel war sich sukzessive von Softwarecode, zu Desktop-

App bis hin zur APK-Datei zu verbessern. Dies war ein Vorhaben, welches über die Anforderungen des Modul hinaus ging. Der Start der Deployment-Strategie stellte die Betrachtung folgender Faktoren dar:

- 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 Möglichkeit, eine App möglichst unkompliziert bereit zu stellen, enttarnte sich als starke Fehleinschätzung. 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, welcher 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 im Abschnitt 5.2.

3 Datensatz

4 Evaluation

4.1 Das Interface/Die App

Die App bietet eine Benutzeroberfläche mit verschiedenen Bildschirmen, wie in den Abbildungen 1, 2, 3, und 4 zu erkennen ist.

In Abbildung 1 ist der Einstiegsbildschirm der App zu sehen. Hier erhalten Benutzer eine kurze Einführung in die App und können die Müdigkeitserkennung starten oder auf die Einstellungen zugreifen. Die eigentliche Müdigkeitserkennung findet im Erkennungsbildschirm (Abbildung 2) statt, wo erkannte Gesichtsmerkmale wie die Augen markiert werden. Bei Erkennung von Müdigkeitsmerkmalen kann die App Warnungen anzeigen oder Alarmtöne abspielen. Im Einstellungsbildschirm (Abbildung 3) können

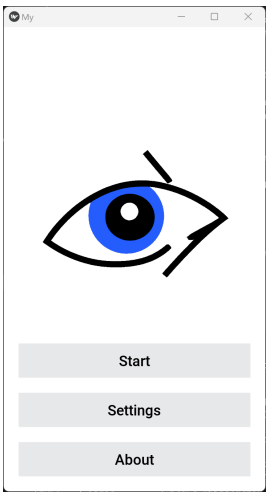

Abbildung 1

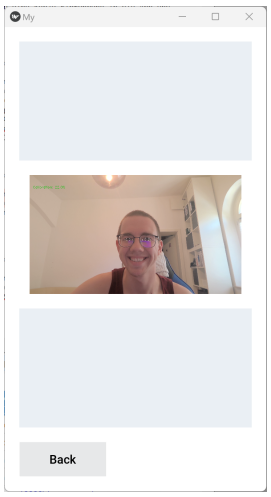

Abbildung 2

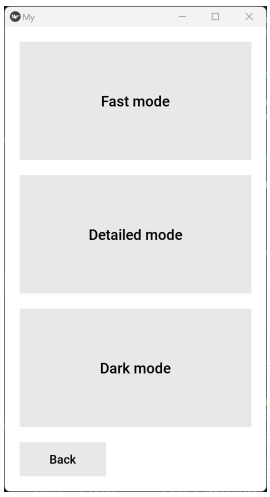

Abbildung 3

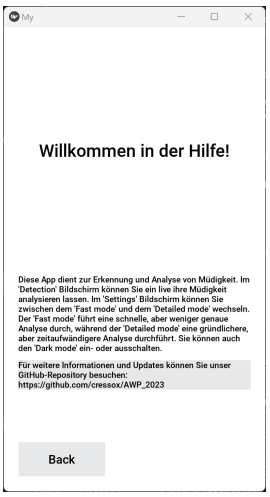

Abbildung 4

Benutzer verschiedene Einstellungen für die Benutzeroberfläche anpassen. Der Hilfebildschirm (Abbildung 4) bietet detaillierte Informationen zur Verwendung der App und ihrer Funktionen.

Grundsätzlich sind wir mit dem Layout zufrieden, da es Minimalismus und Nutzbarkeit vereint. Der gesamte Funktionsumfang wird von der App abgedeckt und dem Nutzer zur Verfügung gestellt. Hierbei ist die Bedienung selbsterklärend und zielführend, ohne die potenziellen Nutzer vor Schwierigkeiten der Orientierung zu stellen. Wie in Abschnitt 5.1 bereits erwähnt, hätte mehr Zeit in die Möglichkeiten der Entwicklung investiert werden können, somit hätte das Design noch interaktiver und flexibler sein können.

5 Lessons Learned

5.1 Herausforderungen in der App-Entwicklung

Eine weitere wichtige Erkenntnis, 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 Projekts erkannt, dass Java und Kotlin als Programmiersprachen besser zu unseren Anforderungen gepasst hätten. Beide Sprachen bieten eine reibungslose Integration der benötigten Bibliotheken wie OpenCV und MediaPipe. Diese Erkenntnis hat uns gezeigt,

wie wichtig es ist, die Flexibilität bei der Wahl der Technologie zu wahren und solche Optionen offen zu halten.

Der nächste wesentliche 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 des Deployments der App. 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 der Benutzer optimal zu erfüllen.

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 ein limitierender Faktor 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, welche in diese Bemühungen investiert wurde, hätte stattdessen genutzt werden können,

um die Potenziale zu erforschen, die bereits im Kapitel 2.5.3 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