Mattis Dietrich, Felix Tischler & Jan-Nicolas Weider

Bericht zum ersten Meilenstein

Arbeitsfortschritt

Für den ersten Meilenstein arbeiteten wir an zwei verschiedenen großen Bereichen, einerseits der Entwicklung von ersten Funktionen sowie Vorarbeiten zum Deployment.

Der erste Teil der Funktionsentwicklung war geprägt von umfangreicher Recherche zu möglichen Vorgehensweisen und auch der Bewertung dieser. Dabei war das Ziel folgende Fragen zu klären:

Wie macht sich Müdigkeit bemerkbar?
Wie detektiere ich ein Gesicht und das Auge?
Wie erkenne ich Landmarks in einem Gesicht?
Wie lässt sich Blinzeln detektieren?
Welche Features zur Detektion von Müdigkeit gibt es?
Wie lässt sich anhand der Features die Müdigkeit detektieren?

Für die Detektion von Müdigkeit im Straßenverkehr gibt es verschiedene Möglichkeiten. Diese lassen sich in drei Ansätze unterteilen (Ramzan et al. [Ramzan2019]):

1. Fahrweise: Aufgrund der Analyse der Fahrweise lässt sich Unaufmerksamkeit durch Müdigkeit ermitteln. Ausschlaggebende Muster sind das Halten (Wach) oder Verlassen (Müde) der Fahrspur.
2. Physiologisch: Über physiologische Daten, wie der Herzfrequenz oder der Atemfrequenz, wird die Müdigkeit detektiert. Eine niedrige Herzfrequenz weist beispielsweise auf einen entspannten, potenziell müden Zustand hin.
3. Verhaltensweise: Das Verhalten eines Fahrers sagt viel über den Müdigkeitszustand aus. So sind Anzeichen für Müdigkeit Gähnen oder ein verändertes Blinzelverhalten.

Bei unserer Arbeit wird das Verhalten des Fahrers analysiert, wobei der Fokus auf dem Auge liegt. Zunächst ist dabei die Detektion des Auges und das Erkennen von Landmarks ein wichtiger Schritt, um darauf folgend Features zu extrahieren. Dabei hängen die Gesichts- und Landmarkdetektion eng zusammen, denn die Wahl des Algorithmus für die Gesichtsdetektion ist abhängig von dem Anwendungszweck der Detektion. In unserem Fall möchten wir Landmarks in einem Gesicht finden, wofür sich der Viola-Jones Algorithmus eignet, welcher nachweislich vielfach verwendet wurde (Ghoddoosian et al. [Gho2019], Arunasalam et al [Aru2020]). Eine gute Implementierung des Viola-Jones Algorithmus bietet der Dlib Face Detector. Dabei wurde der Viola-Jones Algorithmus angepasst und erweitert, um eine Realzeitanwendung zu realisieren. Eine andere Möglichkeit ist, den MediaPipe Face Detector zu nutzen, welcher jedoch schwieriger zu implementieren ist. Ein nicht zu missachtender Vorteil des Dlib Face Detectors ist, dass das Modell vortrainiert wurde und offline anwendbar ist. Das Modell detektiert 68 Landmarks im Gesicht, wovon sich sechs in jeweils einem Auge befinden.

Eine auf den Landmarks aufbauende Metrik, auf welcher Features aufbauen, ist der Wert der Eye Aspect Ratio (EAR). Der Wert berechnet das Verhältnis zwischen den Augenliedern, um die Augenöffnung zu bestimmen. Auch entscheidend ist die Frequenz des Blinzelns (PERCLOS-Wert). Bei Müdigkeit verändert sich das Blinzelverhalten deutlich, was von Dreißig et al. [Dre2020] bereits zur Müdigkeitsdetektion verwendet wurde, auch dies gilt es mit in die Analyse einzubeziehen.

Eine der ursprünglichen Ideen für das Deployment war die Verwendung von Flask als Backend-Server und React Native als Frontend-Framework für die Entwicklung einer Hybrid-App. Hierbei sollte eine REST-API verwendet werden, um die Kommunikation zwischen der App und dem Backend zu ermöglichen.

Flask wurde als Backend-Server gewählt, da es eine einfache und flexible Möglichkeit bietet, eine REST-API zu implementieren und mit der App zu kommunizieren. React Native wurde als Frontend-Framework gewählt, da es eine breite Unterstützung für mobile Plattformen bietet und eine große Entwicklercommunity hat.

Allerdings haben sich die Anforderungen im Laufe der Entwicklung geändert und es ist sehr wahrscheinlich, dass wir kein Machine Learning-Modell mehr verwenden. Dadurch ergeben sich auch andere Anforderungen an das Deployment. Die ursprüngliche Idee mit Flask und React Native wird den neuen Anforderungen voraussichtlich nicht mehr gerecht. Daher werden alternative Ansätze und Technologien in Betracht gezogen, die besser zu den aktuellen Anforderungen passen. Im Laufe der Recherche zum Thema wurden verschiedene Möglichkeiten und Optionen in Betracht gezogen:

1. React als Framework: Mit dem Framework React kann man die Anwendung entwickeln und dabei die WebcamCapture-Komponente nutzen, um auf die Kamera des Geräts zuzugreifen und Videoaufnahmen zu generieren. React bietet eine gute Unterstützung für die Erstellung interaktiver Benutzeroberflächen und ermöglicht es, den Code in Komponenten zu strukturieren.
2. Flask-API: Ursprünglich gab es die Idee, eine Flask-API zu erstellen, um die WebcamCapture-Komponente zu unterstützen und die Datenverarbeitung auf einem Server durchzuführen. Aufgrund Performanceprobleme wurde sich jedoch gegen diese Lösung entschieden.
3. Kivy als Framework: Kivy ist ein Open-Source-Python-Bibliothek zur Entwicklung von Multitouch-Anwendungen und unterstützt sowohl Android als auch iOS. Die Entscheidung für Kivy würde es ermöglichen, den gesamten Code in Python zu schreiben, was den Entwicklungsprozess vereinfachen und die Notwendigkeit eliminieren könnte, eine separate API für die Kommunikation zwischen Frontend und Backend zu schreiben. Zusätzlich könnte die Offline-Fähigkeit von Kivy besonders nützlich für das vorgeschlagene Szenario sein, in dem die Anwendung die Fähigkeit haben sollte, ohne dauerhafte Internetverbindung zu funktionieren.

Erkenntnisse

Wie sich aus der Recherche ergeben hat, sind einige Methoden zur Gesichtsdetektion in der Praxis für Realzeitanwendungen nicht geeignet, da deren Berechnungsaufwand für unseren Anwendungsbereich zu groß ist. Auch gibt es nicht viele Gesichtserkennungsmethoden, welche für die eigene Implementierung genutzt werden können, da viele Gesichtserkennungsmethoden nur käuflich zu erwerben sind. Es gibt einige bis dato unbedachte Schritte zu beachten, insbesondere die Vorverarbeitung der Features. Dies hatten wir in der Planung missachtet, es erscheint jedoch, wie im Arbeitsfortschritt beschrieben, als unerlässlich.

Für das Deployment gilt, dass der Verzicht auf eine API und die Ausführung des Codes clientseitig mit Pyscript viel Potenzial zur Verbesserung der Leistungsfähigkeit bietet. Die direkte Datenverarbeitung auf dem Endgerät ermöglicht eine schnellere Ausführung und reduziert Latenzzeiten. Basierend auf den Erkenntnissen und dem Vergleich von React und Kivy ist nun die Entscheidung zu treffen, ob Pyscript für uns nutzbar ist und wie es am besten in eine App einzubetten ist.

Neben den fachlichen Erkenntnissen ist jedoch auch wichtig anzumerken, dass eine gute Absprache unter den Teammitgliedern Grundlage für ein funktionierendes Projekt ist. Das Deployment und die Anwendung der Detektion an sich beeinflussen sich gegenseitig, somit muss stets in Abstimmung gearbeitet werden. Die direkte Kommunikation beizubehalten ist demzufolge unerlässlich für das weitere Vorgehen und muss somit beibehalten werden.