



**School of
Engineering**

InIT Institut für angewandte
Informationstechnologie

Bachelorarbeit IT11

Talkalyzer: Mobile-App zur automatischen Sprecher-Erkennung

Autoren

Lukas Kündig

Hauptbetreuung

Dr. Mark Cieliebak
Dr. Thilo Stadelmann

Datum

04.06.2014

Erklärung betreffend das selbständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinar massnahmen der Hochschulordnung in Kraft.

Ort, Datum:

Unterschriften:

.....

.....

.....

.....

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Bachelorarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

Derzeit ist keine mobile Applikation in den Applikationsmärkten zu finden, welche eine gute Stimmenerkennung für Personen erreicht und diese dem Benutzer passend präsentiert. Dies könnte jedoch zum Beispiel für ein Meeting oder eine Unterhaltung mit Freunden erwünscht sein, falls herausgefunden werden will, wie viel der eigene Sprecheranteil ausmacht. Mit dieser Arbeit soll untersucht werden, ob ein Algorithmus auf einem Handy implementiert werden kann, welcher eine Person in einer Unterhaltung verlässlich identifizieren und die Sprecheranteile fortlaufend in einem Diagramm der Benutzerin oder dem Benutzer darstellen kann. In dieser Arbeit wurde der Ansatz verfolgt, die Benutzerin oder den Benutzer beim ersten Applikationsstart laut Bilder beschreiben zu lassen, um ihre oder seine Stimme zu erfassen und davon ein Modell zu erstellen. Mit diesem Modell soll anschliessend in einer Unterhaltung die Stimme dieser Person wiedererkannt werden. Ebenfalls soll dargestellt werden, ob jemand anders gesprochen hat oder niemand. Um die Sprechererkennung durchführen zu können wurden bestehende Bibliotheken oder Code-Ausschnitte sowie eigene Implementationen von Komponenten verwendet. Abschliessend soll ausgewertet werden, ob die implementierten Komponenten für ein gutes Resultat genügen. Um andernfalls verbesserte Komponenten einsetzen zu können wurden austauschbare Komponenten definiert, für welche ein Austauschvorgang effizient durchgeführt werden kann. In der Applikation wird die CoMIRVA-Bibliothek verwendet, um die Audiodaten zu Mel-Frequenz-Cepstrum-Koeffizienten (MFCCs) zu verarbeiten. Sind genügend Daten der Stimme erfasst, so wird anschliessend mit diesen Daten ein Sprechermodell mit einem k-Means++-Algorithmus von der Apache Commons Math-Bibliothek und in Java umgeschriebenen C++-Code für den EM-Algorithmus erstellt. Wird eine Unterhaltung mit einem Benutzerprofil gestartet, so werden fortlaufend die zu MFCCs konvertierten Mikrofondaten mit dem Sprechermodell verglichen. Mit einem Schwellenwert wird entschieden, ob die MFCCs eine genug hohe Wahrscheinlichkeit zum Sprechermodell haben, womit die Sprecherbestimmung resultiert. Für die Visualisierung der Sprecheranteile in Diagrammen wird die AChartEngine-Bibliothek verwendet. Die Berechnung eines Sprechermodells variiert je nach Qualität und Menge der Daten. Für die vorausgesetzte Kalibrierungszeit von einer Minute wird das Sprechermodell in etwa fünf Sekunden erstellt. Während einer Unterhaltung werden fortlaufend die eingelesenen Mikrofondaten mit dem Benutzerprofil verglichen. Daraus resultieren die Sprecheranteile, welche in einem sich aktualisierenden Kreisdiagramm dargestellt werden. Mit den implementierten Komponenten war es bei den durchgeführten Tests unter realen Bedingungen jedoch nicht möglich, die Stimme der Benutzerin oder des Benutzers eindeutig zu erkennen. Die erhaltenen Wert für das F-Mass waren im Bereich von 0.41 bis 0.8. Hierfür bedarf es daher weiterer Forschung, um die entscheidenden Komponenten durch verbesserte Verfahren zu ergänzen und dadurch die Erkennungsleistung zu steigern.

Abstract

At the moment, one cannot find a good commercial mobile application which provides good human voice recognition. Such an application may be desired for meetings or conversations with friends in order to find out how big someone's share of the conversation was. The assignment was to examine if an algorithm can be implemented on a mobile phone to reliably identify people in a conversation and continuously illustrate proportional speaking-time in diagram form. In this assignment, the approach was pursued to ask the user to loudly describe pictures at the first application start. This captures the speaker's voice and a speaker model is created. With this model, the speaker should then be recognized in conversation. The software will also recognize if there was another speaker or if no one was speaking at all. In order to conduct the voice recognition, libraries, code cutouts and self-implemented components were used. The final task was to evaluate if the implemented components are sufficient for a good result. To alternatively use enhanced components, exchangeable components were defined. Through this, an exchange process can be realized efficiently. In the application, the CoMIRVA library is used to process the audio data to Mel-frequency cepstral coefficients (MFCCs). When enough data of the voice is captured, a speaker's model is created with a k-Means++-algorithm of the Apache Commons Math library and transcribed into Java C++-Code for the EM-Algorithm. When a conversation with a user profile is started, the data which is captured by the microphone is continuously converted into MFCCs and compared with the speaker's model. A threshold value is used to make a decision whether the MFCCs have a high enough probability of corresponding to the speaker's model which leads to the determination of the speaker. For the visualization of the speaker shares in diagrams, the AChartEngine library is used. The calculation of a model varies according to the quality and the quantity of the captured data. For the assumed calibration time of one minute, the speaker's model will be created in about five seconds. When a conversation is started, the read microphone data is continuously compared with the user's speaker model. That leads to the constantly updating pie-chart diagram which indicates the amount a person has spoken in the whole conversation. With the implemented components, it was not possible to clearly identify a voice under real conditions during the test phase. The resulting f1-score values were in the range from 0.41 to 0.8. Therefore, more research is necessary to complement the essential components through enhanced methods and thus increase the voice identification efficiency.

Inhaltsverzeichnis

1	Einleitung	7
1.1	Motivation	7
1.2	Zielsetzung	7
1.3	Relevante Projekte.....	8
1.4	Terminologie	8
2	Grundlagen	9
2.1	Android.....	9
2.1.1	Main-Thread	9
2.1.2	Activity	9
2.1.3	Service	9
2.1.4	Lebenszyklus	9
2.2	Sprechererkennung.....	9
2.2.1	Mel-Frequenz-Cepstrum-Koeffizient (MFCC)	10
2.2.2	K-Means-Algorithmus.....	10
2.2.3	Gaussian-Mixture-Model (GMM).....	11
3	Konzept und Technologien.....	13
3.1	Konzept	13
3.1.1	Anforderungen an die Applikation.....	13
3.1.2	Hauptaktivitäten aus Sicht der Applikation.....	13
3.1.3	Anwendungsfälle.....	14
3.1.4	Persona	16
3.1.5	GUI.....	17
3.1.6	Sprechererkennung.....	19
3.1.7	Codestruktur	21
3.2	Potentiell relevante Technologien	22
3.2.1	Diagramme	22
3.2.2	Sprechererkennung.....	22
3.3	Verwendete Technologien.....	23
3.3.1	Diagramme	23
3.3.2	Sprechererkennung.....	23
3.4	Verworfenen Technologien	24
3.4.1	Diagramme	24
3.4.2	Spracherkennung	25
4	Resultat.....	26
4.1	GUI.....	26
4.1.1	Kalibrierung.....	26
4.1.2	Einstiegsseite	28
4.1.3	Unterhaltung führen.....	29
4.1.4	Kalibrierung erweitern.....	30

4.1.5	Diagramme	30
4.2	Codestruktur	31
4.2.1	Kalibrierung.....	35
4.2.2	Unterhaltung führen.....	36
4.2.3	Kalibrierung erweitern.....	38
4.2.4	Austauschbare Komponente	39
4.3	Sprechererkennung.....	40
5	Evaluation.....	44
5.1	GUI.....	44
5.1.1	Vorgehen	44
5.1.2	Ergebnisse	44
5.2	Codestruktur	45
5.3	Sprechererkennung.....	47
5.3.1	Evaluierung der MFCCs, der k-Means- und der Sprechermodell-Implementation.....	48
5.3.2	Evaluierung der Qualität der Sprechererkennung	52
6	Zusammenfassung und Ausblick.....	59
6.1	Zusammenfassung.....	59
6.2	Taktischer Ausblick.....	59
6.2.1	Fehler in der Software	59
6.2.2	GUI.....	60
6.2.3	Codestruktur	61
6.2.4	Sprechererkennung.....	61
6.3	Strategischer Ausblick.....	62
6.3.1	Sprechererkennung.....	62
7	Verzeichnisse.....	63
7.1	Literaturverzeichnis.....	63
7.2	Abbildungsverzeichnis	66
7.3	Diagramme	66
7.4	Tabellen	67
8	Anhang	69
8.1	Projektmanagement	69
8.1.1	Offizielle Aufgabenstellung	69
8.1.2	Aufgabenstellung Bachelorarbeit Stampfli	70
8.2	Weiteres.....	71
8.2.1	Auszug aus dem Mailverkehr über den Ersatz der AudioInputStream-Klasse mit Herr Klaus Seyerlehner.....	71
8.2.2	Auszug aus dem Mailverkehr bezüglich Copyright [10].....	71

1 Einleitung

In dieser Projektarbeit soll eine Android-Applikation entwickelt werden, welche dem Benutzer¹ auf eine verständliche Art vermittelt, wieviel während einer Unterhaltung er, jemand anders oder niemand gesprochen hat. Dabei sollen dem Benutzer fortlaufend die aktuellsten Sprecheranteile dargestellt werden.

Um feststellen zu können, ob der Benutzer oder jemand anders spricht, wird ein Modell von der Stimme des Benutzers benötigt. Deshalb wird vorausgesetzt, dass beim ersten Start der Applikation die Stimme des Benutzers erfasst wird. Anhand der erfassten Daten wird anschliessend ein Sprechermodell erstellt, welches dem Benutzerprofil hinterlegt wird. Anschliessend kann der Benutzer eine Unterhaltung starten und diese von der Applikation analysieren lassen. Pausiert er eine Unterhaltung, so werden dem Benutzer weitere Möglichkeiten geboten, die Unterhaltung genauer auswerten zu können.

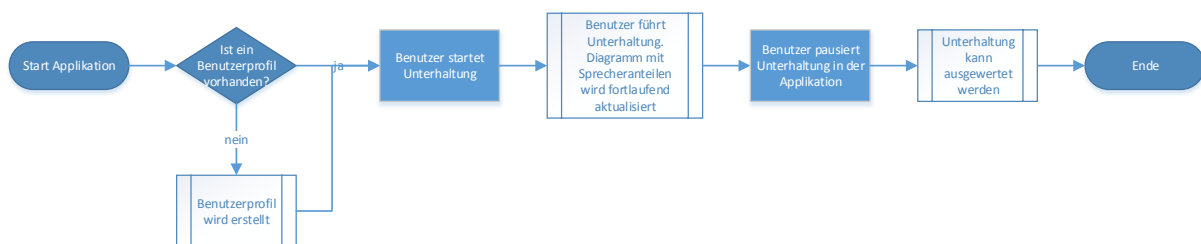


Diagramm 1 - Ablauf der Applikation

1.1 Motivation

Im Alltag ist man in viele Gespräche verwickelt. Sei es zum Beispiel bei der Arbeit, Zuhause beim Essen oder bei einem Gespräch in einer Bar. Dies können Gespräche mit vertrauten Personen oder mit bisher fremden Personen sein. Auch das Thema ist oft ein anderes. Wie wäre es also, sich selber zu analysieren, wie man sich in den verschiedenen Gesprächen verhält? Eventuell verfolgt man das Ziel, nicht mehr als andere zu sprechen oder nur in bestimmten Zeitabschnitten. Es kann auch sein, dass aus reiner Neugier seine eigene Gesprächszeit ermitteln werden will.

Die Talkalyzer-Applikation, soll eine Unterhaltung analysieren und erkennen, welcher Sprecher, dass spricht. Es existiert bereits eine ähnliche Applikation namens Talk-o-meter [1], welche jedoch keine zufriedenstellenden Ergebnisse liefert [1]. Kommt die Talkalyzer-Applikation in den Android-Market und findet bei den ersten Benutzern Gefallen, so lässt sich eine schnelle Verbreitung erhoffen.

Voraussetzung dafür ist eine gut durchdachte Applikation, welche eine breite Benutzergruppe anspricht und gute Resultate liefert. Ebenfalls spielt die Performance eine Herausforderung, denn ein Sprechererkennungsalgorithmus, welcher eingesetzt werden muss, hat auf einem Handy nur sehr begrenzt Ressourcen zur Verfügung.

1.2 Zielsetzung

Die Zielsetzung gliedert sich in drei Bereiche. Zum einen ist das Ziel der Arbeit, eine praktische Applikation für den Benutzer zu entwickeln, welche er gerne verwendet. Dazu gehört eine übersichtliche und verständliche Benutzeroberfläche.

Ebenfalls sollen Komponenten für die Sprechererkennung eingesetzt werden, mit welchen gute Resultate erzielt werden. Die Sprechererkennung soll dabei unter realen Bedingungen getestet werden, so dass ein Fazit gemacht werden kann, ob die Applikation die Erwartungen eines Benutzers erfüllen kann.

¹ Um die Darstellung einfach zu halten, wird im Dokument die weibliche Form nicht erwähnt. In jedem Fall ist jedoch auch die weibliche Person damit gemeint.

Für einen Applikationsentwickler sollen einfach austauschbare Komponenten definiert werden, bei welchen durch wenige Schritte ermöglicht wird, sie zu ersetzen ohne den Ablauf oder die Funktionalität der gesamten Applikation damit zu beeinflussen.

1.3 Relevante Projekte

Vor dieser Arbeit existierte das Projekt Talkalyzer bereits. Es wurde Matlab-Code geschrieben, welcher Sprechermodelle und Visualisierungen dazu erstellt. Für die Sprechermodelle wird die Voicebox-Bibliothek [2] verwendet. Ebenfalls wurde auf Android eine Applikation entwickelt, welche die Erstellung des Sprechermodells auf einfache Art mit der CoMIRVA-Bibliothek [3] durchgeführt. Es stellte sich jedoch heraus, dass die Ergebnisse nicht zufriedenstellend sind. Gleichzeitig wird zu dieser Bachelorarbeit eine weitere Bachelorarbeit durchgeführt, welche das Ziel hat, verbesserte Resultate für die Sprechererkennung zu entwickeln.

1.4 Terminologie

Nachfolgend werden Begriffe erklärt, die für die vorliegende Arbeit von zentraler Bedeutung sind.

Begriff	Beschreibung
GUI	Das GUI definiert die grafische Benutzeroberfläche, welche sich aus grafischen Elementen und Steuerelementen zusammensetzt. Es erlaubt einem Benutzer Interaktionen mit der Maschine vorzunehmen und so die Applikation zu steuern.
Audiodaten	Audiodaten werden jene Daten genannt, welche vom Mikrofon des Handys eingelesen wurden.
Feature	Features werden verwendet, um Merkmale von der Stimme einer Person zu repräsentieren.
Kalibrierung	Kalibrierung wird jener Abschnitt genannt, in welchem die Stimme bzw. die Features des Benutzers erfasst und ein Modell daraus erstellt wird, inklusive der dafür notwendigen Vorbereitungen und Nacharbeiten.

Tabelle 1 – Terminologie

2 Grundlagen

Nachfolgend werden Komponenten und Begriffe, welche in dieser Arbeit eingesetzt werden, erläutert. Zum einen werden wenige wichtige Komponenten von Android beschrieben, da die Applikation auf Android entwickelt wird. Zum anderen werden die Grundlagen für die Sprechererkennung erklärt.

2.1 Android

Die Applikation wird für das Android-Betriebssystem von Google entwickelt. Eine Applikation besteht aus Java-Code, welcher unter anderem das Verhalten von Android definiert und diversen XML-Dateien, mit welchen das GUI gebildet wird. Als geeignete Entwicklungsumgebung wird von Google „Android-Studio“ [4] angeboten.

2.1.1 Main-Thread

Der Main-Thread repräsentiert denjenigen Prozess bzw. Thread, in welchem die Applikation gestartet wird. Er wird manchmal auch UI-Thread genannt [5], da er standardmässig für das GUI zuständig ist. Wird er mit rechenintensiven Aufgaben belegt, so führt dies zu Verzögerungen beim GUI.

2.1.2 Activity

Eine Activity ist für das Laden des GUIs und die Interaktion mit dem Benutzer zuständig. Typischerweise stellt sie eine eigenständige Tätigkeit dar [6]. Sie besteht aus einer Java-Klasse mit welcher das Verhalten der Activity selbst sowie die Interaktionen mit dem Benutzer definiert werden und einer XML-Datei, welche für das GUI zuständig ist.

2.1.3 Service

Ein Service repräsentiert eine Komponente einer Applikation, in welcher Code ausgeführt wird, der sich nicht auf Benutzerinteraktionen ausrichtet. Er bietet deshalb kein GUI. Ein Service wird etwa von einer Activity gestartet. Eine Activity, welche einen Service aufruft, kann mit dem Service anhand eines Bindings [7] Service Methoden ausführen. In einem vom Service erzeugten Thread wird rechenintensive Arbeit erledigt, welche den Main-Thread nicht blockiert.

2.1.4 Lebenszyklus

Activities und Services besitzen je einen eigenen Lebenszyklus [6] [8]. Dieser definiert einen vordefinierten Ablauf, wann die Komponente wie verwaltet wird. Dabei kann jeder Zustand selbst mit Code ergänzt werden.

2.2 Sprechererkennung

Um einen Sprecher in einer Unterhaltung erkennen zu können, wird ein Modell von seiner Stimme benötigt. Ein Modell wird beim ersten Start der Applikation erstellt. Bei diesem Vorgang werden die Mikrofondaten so verarbeitet, dass die Stimmmerkmale des Benutzers erfasst werden. Diese Stimmmerkmale werden in dieser Arbeit mit MFCCs (siehe Kapitel 2.2.1) festgehalten. Sind genügend Daten der Stimme des Benutzers erfasst worden, so kann mit den daraus erstellten MFCCs ein Modell trainiert und anschliessend dem Benutzer hinterlegt werden.

Startet der Benutzer mit seinem Profil eine Unterhaltung, welche er von der Talkalyzer-Applikation analysieren lassen möchte, so werden wieder die Mikrofondaten zu MFCCs konvertiert und diese mit dem Sprechermodell des Benutzers verglichen bzw. getestet, woraus eine Wahrscheinlichkeit für die

Ähnlichkeit von den Features zu seinem Modell berechnet wird. Ist diese Wahrscheinlichkeit genug hoch, so wird entschieden, ob der Sprecher mit dem Sprecherprofil gesprochen hat.

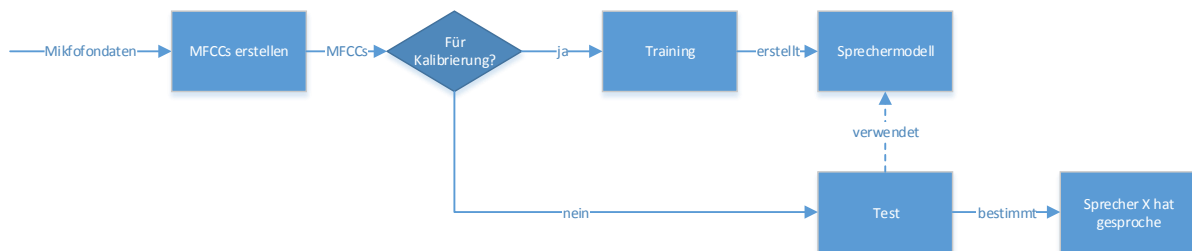


Diagramm 2 - Ablauf Sprechererkennung

2.2.1 Mel-Frequenz-Cepstrum-Koeffizient (MFCC)

Mel-Frequenz-Cepstrum-Koeffizienten werden aus Audiodaten generiert und als Features verwendet. Um ein MFCC zu generieren wird in einem bestimmten Zeitraum durch mathematische Umformungen des Frequenzspektrums, dieses in geeigneter Form für eine automatische Weiterverarbeitung umgeformt, etwa als Array mit einer bestimmten Anzahl an Dimensionen. MFCCs werden verwendet, um die Stimme einer Person in einem bestimmten Zeitraum abzubilden und zu charakterisieren. Soll die Lautstärke in den MFCCs miteinbezogen werden, so ist sie in der ersten Dimension enthalten. Aus einem MFCC kann die ursprüngliche Sprache wiederhergestellt werden.

2.2.2 K-Means-Algorithmus

Der k-Means-Algorithmus dient zum Klassifizieren von Elementen. Dabei ist das Ziel, ähnliche Elemente in einer Gruppe zu haben und die Ähnlichkeit zwischen den Gruppen klein zu halten. Im Falle dieser Arbeit wird mit dem k-Means-Algorithmus angestrebt, MFCCs so zu gruppieren, dass jede Gruppierung von MFCCs eine bestimmte Eigenschaft der Stimme eines Sprechers repräsentiert. Nachfolgende Abbildung zeigt ein Beispiel, wie ungruppierte MFCCs einer Gruppe, welche je durch eine Farbe dargestellt wird, zugeordnet werden.

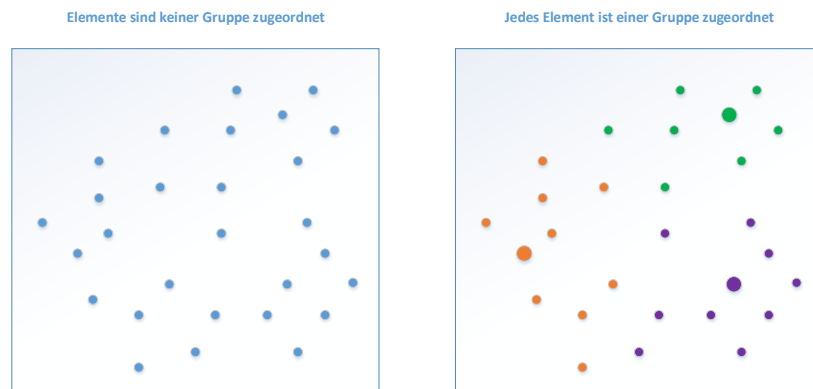


Abbildung 1 - Visualisierung für die Zuordnung von Elementen in eine Gruppe anhand von Schwerpunkten

Der eingesetzte k-Means++-Algorithmus wählt anfangs ein zufälliges Element als Schwerpunkt für die erste Gruppe aus. Anschliessend wird die Distanz von jedem Element zum nächstgelegenen Schwerpunkt berechnet. Der nächste Schwerpunkt, welcher bestimmt wird, wird anschliessend so gewählt, dass er die weiteste Distanz im Quadrat zu den bisher bestimmten Gruppenschwerpunkten aufweist [9]. Dies wird solange durchgeführt, bis eine bestimmte Anzahl Gruppenschwerpunkte definiert wurden. Diese Initialisierungsmethode ist gegenüber dem k-Means-Algorithmus fast doppelt so schnell [9].

Sind alle Schwerpunkte bestimmt, so werden die Elemente jener Gruppe zugeordnet, von welcher der Gruppenschwerpunkt ihnen jeweils am nächsten ist. Mit den Elementen jeder Gruppe wird anschliessend der Schwerpunkt neu berechnet. Mit den neu berechneten Schwerpunkten wird die Zuweisung der

ihnen am nächsten stehenden Elemente erneut durchgeführt. Die Berechnung von neuen Schwerpunkten und dessen Zuweisungen von Elementen wird für eine bestimmte Anzahl Durchgänge durchgeführt.

Aus der Berechnung des k-Means-Algorithmus werden die gebildeten Gruppen mit den jeweils enthaltenen Elementen erhalten. Für diese Projektarbeit werden nachfolgende Eigenschaften benötigt, welche aus dem Resultat des k-Means-Algorithmus abgeleitet werden können.

Legende	
K_i	Anzahl MFCCs in Gruppe i.
N	Anzahl MFCCs insgesamt.
$x_{j,d}$	Wert der Dimension d von einem MFCC in der Gruppe i.

Tabelle 2 - Legende für Tabelle 3

Eigenschaft	Beschreibung
Gewicht	Jede Gruppe erhält das Gewicht anhand der Anzahl ihren zugeteilten Elementen, geteilt durch die Gesamtanzahl von Elementen. Die Resultate werden in einem eindimensionalen Array abgespeichert. $w_i = \frac{K_i}{N}$
Durchschnitt	In jeder Gruppe wird für jede Dimension der Durchschnitt bzw. Erwartungswert berechnet. Dazu wird für jede Dimension anhand aller Elemente pro Gruppe der Durchschnitt berechnet. Die Werte werden in einem zweidimensionalen Array abgespeichert. $\mu_{i,d} = \frac{1}{K_i} * \sum_{j=1}^{K_i} x_{j,d}$
Varianz	Die Varianz wird pro Gruppe und Dimension berechnet und in einem zweidimensionalen Array gespeichert. $\sigma_{i,d}^2 = \frac{1}{K_i - 1} * \sum_{j=1}^{K_i} (x_{j,d} - \mu_d)^2$

Tabelle 3 – Benötigte Berechnungen anhand dem Resultat des k-Means-Algorithmus

Die Erwartungswerte und die Varianzen bilden die Normalverteilungen, aus welchen anschliessend das Gaussian-Mixture-Model berechnet wird.

2.2.3 Gaussian-Mixture-Model (GMM)

Ein Gaussian-Mixture-Model besteht aus mehreren gewichteten Normalverteilungen. Diese werden in dieser Arbeit mit dem k-Means-Algorithmus erhalten. Das Modell stellt dabei die gewichtete Summe dieser Normalverteilungen dar. Abgebildet ist ein Beispiel eines Gaussian-Mixture-Modells mit zwei Komponenten bzw. zwei Normalverteilungen.

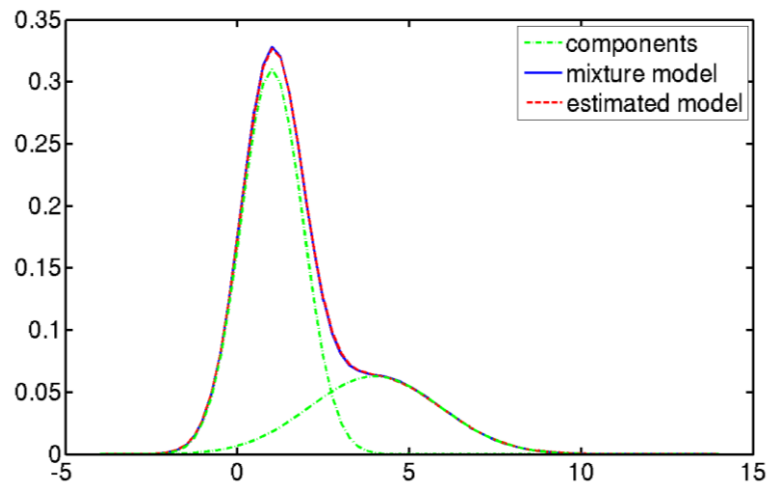


Abbildung 2 - Gaussian-Mixture-Model (Bildquelle: [10])

Ein GMM wird in dieser Arbeit verwendet, um die erfassten Stimmdaten eines Sprechers als MFCCs in geeigneter Weise zu speichern. Grundsätzlich lässt sich ein GMM durch einen bestimmten Algorithmus, etwa dem Expectation-Maximization-Algorithmus für die Genauigkeit des Modells trainieren. Durch eine weitere Funktion, der Log-Likelihood-Methode, wird eine Wahrscheinlichkeit berechnet, wie ähnlich bestimmte MFCCs zum Modell sind.

Expectation-Maximization-Algorithmus (EM-Algorithmus)

Da im GMM nicht die gesamte Stimme des Sprechers erfasst ist, müssen Annahmen zu den fehlenden Werten gemacht werden, welche die Stimme des Sprechers ebenfalls repräsentieren würden. Es muss deshalb ein neues Modell erstellt werden, welches die fehlenden Werte schätzt.

Als Ausgangslage wird das Gaussian-Mixture-Model genommen, welches mit den Werten des k-Means-Algorithmus initialisiert wurde. Es werden im EM-Algorithmus grundsätzlich zwei Schritte durchgeführt, der Expectation- und der Maximization-Schritt.

Der Expectation-Schritt besteht darin, einen Erwartungswert für die fehlenden Werte bzw. deren gesuchte Verteilung zu berechnen. Der zweite Schritt bestimmt mit der Maximum-Likelihood-Methode [11] die gesuchten Werte.

Diese zwei Schritte werden solange wiederholt, bis eine gewisse Grenze erreicht ist, so dass keine grosse Veränderung am geschätzten Modell mehr vorgenommen werden kann.

Log-Likelihood-Methode

Die Log-Likelihood-Funktion ist die logarithmierte Likelihood-Funktion. Mit der Log-Likelihood-Funktion wird die Wahrscheinlichkeit von einem Feature zum Modell berechnet. Diese Methode ist ebenfalls im Code des Sprechermodells vorhanden. Die Werte der Wahrscheinlichkeit sind dabei meist negativ, können jedoch auch positiv resultieren. Je höher die Werte sind, desto wahrscheinlicher ist, dass ein Feature zum Modell passt.

3 Konzept und Technologien

In diesem Kapitel werden die Bestandteile, das Vorgehen und die geplante Umsetzung dieser Arbeit beschrieben sowie Technologien bezüglich ihrer Eignung ausgewertet und analysiert.

3.1 Konzept

Für die Arbeit sind nachfolgende Eigenschaften definiert. Anhand von diesen wird die Applikation entwickelt und getestet.

Eigenschaft	
Betriebssystem	Android.
Zielgerät	Samsung Galaxy S 2.
Minimum API-Level	16 [12].
Abhängigkeiten	Funktionierendes Mikrofon am Handy vorhanden.

Tabelle 4 – Applikationseigenschaften

3.1.1 Anforderungen an die Applikation

Ein Benutzer hat das Ziel, ein Gespräch mit der Applikation zu analysieren und Informationen über die Sprecheranteile zu erhalten. In jedem Fall muss zuerst ein Sprecherprofil erstellt werden. Es wird angenommen, dass ein Benutzer für maximal eine Minute bereit ist, Aufwand für den Erstgebrauch der Applikation zu leisten. Das Ziel der Applikation ist, dass möglichst viele Personen die Applikation verstehen und bedienen können. Sie sollte deshalb geringe Anforderungen an den Benutzer enthalten. Dies heisst zum Beispiel, dass der Ablauf klar sein muss und die dargestellten Informationen verständlich sind. Der Sprechererkennungsalgorithmus sollte für den Benutzer gute Resultate liefern. Es ist jedoch nicht das Ziel der Arbeit, einen Algorithmus zu verbessern oder zu erforschen. Es wird deshalb vorausgesetzt, dass einzelne oder mehrere Komponenten der Sprechererkennung nach Abschluss dieser Bachelorarbeit einfach ausgetauscht werden können.

3.1.2 Hauptaktivitäten aus Sicht der Applikation

Von der Applikation werden grundsätzlich vier Hauptaktivitäten ausgeführt.

Aktivität	Beschreibung
Mikrofondaten einlesen und zu MFCCs verarbeiten	Während der Kalibrierung oder einer Unterhaltung werden kontinuierlich Mikrofondaten eingelesen. Diese müssen danach für das Sprechermodell als MFCCs zur Verfügung stehen.
Sprechermodell erstellen	Anhand der eingelesenen Audiodaten eines Benutzers wird ein Sprechermodell erstellt, welches die Stimme des Sprechers modelliert. Ein bestehendes Sprechermodell soll mit weiteren Audiodaten der Stimme des Benutzers erweitert werden können.
Sprechererkennung ausführen	In der Applikation existiert eine Entscheidungslogik, welche fortlaufend MFCCs mit dem Sprechermodell vergleicht, um festzustellen, ob der Sprecher mit dem Sprecherprofil gesprochen hat.
Diagramme aktualisieren	Die ausgewerteten Daten der Sprechererkennung werden dem Benutzer gemäss Tabelle 11 in Diagrammen dargestellt.

Tabelle 5 - Hauptaktivitäten der Applikation

3.1.3 Anwendungsfälle

Für die Applikation gibt es drei Hauptanwendungsfälle. Sie werden ebenfalls verwendet, um die Unterkapitel für das GUI, die Codestruktur oder die Sprechererkennung zu unterteilen.

Anwendungsfall 1.01

Kennzeichnung	Anwendungsfall 1.01
Beschreibung	Der Benutzer startet die Applikation das erste Mal. Es wird die Kalibrierung gestartet.
Beteiligte Akteure	Benutzer und Applikation.
Vorbedingung	Applikation wurde noch nie gestartet. Keine Applikationsdaten sind vorhanden.
Invarianten	Keine.
Nachbedingung und Ergebnis	Benutzer ist kalibriert. Ein Modell seiner Stimme wurde erstellt und ist im Profil des Benutzers gespeichert. Das Profil kann für Unterhaltungen verwendet werden.
Standardablauf	
Benutzer	Applikation
1. Startet die Applikation. 3. Gibt einen Profilnamen ein. 5. Beschreibt Bilder.	2. Fordert den Benutzer auf, ein Profil anzulegen. 4. Erstellt ein Profil mit eingegebenem Namen. Fordert den Benutzer auf, laut Bilder zu beschreiben. 6. Erfasst die Stimme des Benutzers. 7. Erstellt ein Sprechermodell aus den erfassten Audiodaten. 8. Zeigt dem Benutzer die standardmässige Einstiegsseite an.
Alternativer Ablauf	
Benutzer	Applikation
1. Startet die Applikation. 3. Verlässt die Applikation.	2. Fordert den Benutzer auf, ein Profil anzulegen. 4. Pausiert die Kalibrierung. Wird die Kalibrierung später nicht abgeschlossen, werden die Daten verworfen und beim nächsten Applikationsstart erneut mit der Kalibrierung begonnen.

Tabelle 6 - Anwendungsfall 1.01

Anwendungsfall 1.02

Kennzeichnung	Anwendungsfall 1.02
Beschreibung	Der Benutzer startet eine Unterhaltung.
Beteiligte Akteure	Benutzer, Applikation und Betriebssystem.
Vorbedingung	Applikation ist gestartet und es existiert mindestens ein kalibriertes Benutzerprofil.
Invarianten	Sprechermodell wird nicht verändert.
Nachbedingung und Ergebnis	Gestartete Unterhaltung wird analysiert und ausgewertet.

Standardablauf	
Benutzer	Applikation
1. Startet die Applikation.	
2. Startet eine Unterhaltung mit einem ausgewählten Profil.	3. Erfasst und wertet die Audiodaten aus. Stellt die ausgewerteten Daten grafisch dar.
4. Pausiert die Unterhaltung.	5. Detailliertere Informationen werden aufbereitet und dem Benutzer zugänglich gemacht.
6. Beendet die Unterhaltung.	7. Verwirft alle erstellten Daten. Das Speichern einer Unterhaltung wird in Kapitel 6.2.2 thematisiert.

Tabelle 7 - Anwendungsfall 1.02

Anwendungsfall 1.03

Kennzeichnung	Anwendungsfall 1.03
Beschreibung	Der Benutzer erweitert die Kalibrierung für sein Sprecherprofil.
Beteiligte Akteure	Benutzer und Applikation.
Vorbedingung	Applikation ist gestartet und der Benutzer hat die Option für das Erweitern der Kalibrierung seines Benutzerprofils ausgewählt. Für das ausgewählte Profil existieren bereits Kalibrierungsdaten.
Invarianten	Keine bisherigen Kalibrierungsdaten werden überschrieben.
Nachbedingung und Ergebnis	Neues Sprechermodell wurde mit den bisherigen und den neu eingelesenen Audiodaten erstellt und im Benutzerprofil abgespeichert. Bisheriges Modell wird verworfen.
Standardablauf	
Benutzer	Applikation
1. Wählt die Option aus, um die Kalibrierung für sein Profil zu erweitern.	
2. Beschreibt Bilder.	3. Erfasst Audiodaten.
4. Beendet die Kalibrierung.	5. Berechnet neues Modell aus den bestehenden und den neuen Kalibrierungsdaten.
	5. Verwendet neues Modell für das Sprecherprofil.
	6. Zeigt dem Benutzer die standardmässige Einstiegsseite an.
Alternativer Ablauf	
Benutzer	Applikation
1. Wählt die Option aus, um die Kalibrierung für sein Profil zu erweitern.	
2. Beschreibt Bilder.	3. Erfasst Audiodaten.
4. Beendet die Kalibrierung ohne zu speichern.	5. Verwirft neu erstellte Daten. Bisheriges Sprechermodell bleibt bestehen.

Tabelle 8 - Anwendungsfall 1.03

3.1.4 Persona

Eine Persona definiert eine zukünftige Benutzergruppe anhand eines fiktiven Benutzers. Sie konkretisiert die Eigenschaften und die Anforderungen sowie das Nutzungsverhalten dieser Gruppe bezüglich der Applikation. Für die Talkalyzer-Applikation sind nachfolgend zwei Personas beschrieben.

Persona „Felix Burkhalter“

	Felix Burkhalter «Ich will wissen, wie viel ich an den Bewerbungsgesprächen spreche» Vor einem halben Jahr hat sich Felix ein neues Android-Handy gekauft. Er trägt sein Handy seither jederzeit bei sich, denn er weiss viel anzufangen damit. Etwa hat er viele nützliche Applikationen installiert, welche ihm seinen Alltag erleichtern. Wenn er eine neue Applikation das erste Mal startet, will er sich sofort zurechtfinden und wissen, wie er mit der Applikation umgehen muss. Zurzeit muss er viele Bewerbungsgespräche führen. Bei diesen Bewerbungsgesprächen wüsste er gerne, wie viel er und wie viel sein Bewerber beim Gespräch jeweils spricht.	Anforderungen: - Fortlaufende Aktualisierung der Sprecheranteile - Verständliche Visualisierung der Sprecheranteile Frustrationen: - Applikationsabstürze - Unklarer Ablauf der Applikation
Männlich 42 Jahre Verheiratet, 2 Kinder Chef von Werbeagentur Installiert immer wieder eine neue Handy-Applikation, in welcher er einen Nutzen für sich erkennen kann und probiert diese aus.		

Abbildung 3 - Persona Felix Burkhalter (Bildquelle: [13])

Persona „Selina Meier“

	Selina Meier «Ich will eine Unterhaltung mit meiner Kollegin analysieren» Selina lädt viel ihre Kollegin Sandra zu sich nach Hause ein. Sie kochen sich dann etwas Feines und unterhalten sich dabei über ihren Alltag. Nun ist Selina auf die Idee gekommen, ihr Gespräch mit einer Handy-Applikation analysieren zu lassen. Sie ist neugierig, ob sie oder Sandra jeweils mehr zum Erzählen hat.	Anforderungen: - Einfache Bedienung der Applikation - Visualisierung der Sprecheranteile Frustrationen: - Applikationsabstürze - Unklarer Ablauf der Applikation
Weiblich 26 Jahre Single Dentalassistentin Geniesst die Gespräche mit ihrer Kollegin.		

Abbildung 4 - Persona Selina Meier (Bildquelle: [14])

3.1.5 GUI

Für das GUI der Applikation werden nachfolgende Kriterien als entscheidend definiert.

Kriterium	Beschreibung	Überprüfbarkeit
Aufgabenangemessenheit	Die Funktionen und Interaktionen beschränken sich auf die Nötigsten.	Ein grafisches Element wird nur dann dargestellt, falls es im Zusammenhang mit der präsentierten Aktivität steht.
Erwartungskonformität	Verwendete GUI-Elemente sind einheitlich und für den Benutzer verständlich. Der verwendete Text ist in Englisch verfasst.	Der Benutzer versteht den Sinn oder die Auswirkung der eingesetzten Elemente und hindern ihn somit nicht, das Programm zu bedienen. Der verwendete Text ist in Englisch und für den Benutzer verständlich.
Selbstbeschreibungsfähigkeit	Dem Benutzer wird durch GUI-Elemente bzw. Rückmeldungen verständlich gemacht, was die Applikation ausführt.	Der Benutzer weiss stets, was die Applikation ausführt oder ausgeführt hat.
Lernförderlichkeit	Die verwendeten Elemente sowie der Ablauf sind für den Benutzer verständlich oder in kurzer Zeit erlernbar.	Die verwendeten Elemente oder Interaktionen sind für den Benutzer verständlich, so dass er die Applikation ohne lange zu überlegen bedienen kann. Ist ein Element oder eine Interaktion nicht sofort verständlich, so wird dem Benutzer nach kurzer Zeit klar, wie damit Umzugehen ist.

Tabelle 9 - Kriterien für das GUI

Kalibrierung

Beim ersten Start der Applikation wird zuerst eine Kalibrierung gestartet. Dem Benutzer wird die Erklärung dafür mit Text beschrieben. Hierfür wird Text gewählt, da angenommen wird, mit erklärenden Bildern den Benutzer nicht gleich verständlich und einfach informieren zu können. Um vom Benutzer die benötigten Sprachdaten zu erhalten wurden drei Methoden untersucht. Diese sind nicht nur fürs GUI relevant, sondern können die Qualität der erfassten Daten bei der Kalibrierung und somit die Resultate des Sprechererkennungsalgorithmus ebenfalls beeinflussen.

Telefongespräch analysieren	
Vorgehen	Beim nächsten Telefonanruf, welcher der Benutzer tätigt, wird die Stimme des Benutzers von der Applikation analysiert.
Vorteile	Die Person spricht sehr wahrscheinlich ähnlich wie in einem Gespräch, welches sie mit der Applikation analysieren möchte. Es können durchaus viele Sprachdaten des Benutzers erfasst werden, falls ein langes Telefongespräch geführt wird und der Benutzer viel spricht.
Nachteile	Die Person wird gezwungen, einen Telefonanruf zu tätigen. Ohne kann die Applikation nicht benutzt werden. Einerseits können dadurch Kosten entstehen, andererseits muss die Person eine Unterhaltung führen, welche sie sonst vielleicht in diesem Moment nicht geführt hätte.
Text vorlesen	
Vorgehen	Einem Benutzer wird ein Text angezeigt, welchen er laut ablesen soll.
Vorteile	Die Person führt eine einfache Aufgabe durch.

Nachteile	Einerseits muss dem Benutzer ein Text in seiner Sprache vorliegen, andererseits kann es für den Benutzer langweilig sein, einen Text abzulesen. Des Weiteren ist das Ablesen eines Textes nicht unbedingt authentisch mit dem Sprechen in einer Unterhaltung.
Bild beschreiben	
Vorgehen	Dem Benutzer werden Bilder gezeigt, welche er laut beschreiben soll.
Vorteile	Bilder beschreiben ist unabhängig von einer bestimmten Sprache. Zudem kann ein Bild sehen und beschreiben unterhaltenden sein. Die Person sieht ausserdem sofort, falls ihr das Bild nicht zusagt oder nichts dazu einfällt und kann das nächste Bild auswählen.
Nachteile	Es wird eine gewisse Kreativität vorausgesetzt, Bilder ausführlich beschreiben zu können. Es müssen für den Benutzer genügend Bilder zum Beschreiben verfügbar sein.

Tabelle 10 – Untersuchte Kalibrierungsmethoden

Die ausgewählte Methode ist das Beschreiben der Fotos. Es wird als die benutzerfreundlichste Art angesehen, vom Benutzer effizient und möglichst authentische Sprachdaten zu erhalten.

Unterhaltung führen

Während dem Gespräch sollen dem Benutzer die aktuellen Sprecheranteile dargestellt werden. Dabei wird als aktuell betrachtet, was der Benutzer als Aktualisierungsintervall für die Diagramme in den Einstellungen der Applikation ausgewählt hat. Des Weiteren wird angenommen, dass der Benutzer während der Unterhaltung keine detaillierte Auswertung sehen möchte, da er primär mit dem Unterhalten beschäftigt ist. Das Ziel ist, dass eine Bildschirmseite für die Unterhaltung verwendet wird.

Unterhaltung auswerten

Hat ein Benutzer die Unterhaltung pausiert, so soll ihm einerseits ein Diagramm dargestellt werden, welches den Anteil jedes Sprechers zeigt und ein Diagramm, welches den Zeitverlauf darstellt. Zudem soll es dem Benutzer möglich sein, Daten für einen bestimmten Zeitraum grafisch dargestellt zu bekommen. Dafür soll eine Einstellmöglichkeit implementiert werden, mit welcher ein Start- und Endzeitpunkt für die darzustellenden Daten eingestellt werden kann.

Kalibrierung erweitern

Das Erweitern der Kalibrierung soll mit denjenigen Komponenten durchgeführt werden, welche auch für die Kalibrierung verwendet werden.

Diagramme

Während einer Unterhaltung wird fortlaufend entschieden, welcher Sprecher wann gesprochen hat. Diese Information dient dem Benutzer für das Analysieren des Gesprächs und muss dem Benutzer in den Diagrammen auf eine verständliche Art veranschaulicht werden. Dabei spielen verschiedene Faktoren eine Rolle.

Eigenschaft	Begründung	Konzept
Gesamter Anteil jedes Sprechers sichtbar	Der Benutzer will wissen, ob er mehr, weniger oder gleich viel wie andere Gesprächsteilnehmer gesprochen hat.	Dem Benutzer wird ein Diagramm dargestellt, welches die prozentualen Sprecheranteile der gesamten Unterhaltung darstellt.
Anteil jedes Sprechers im Zeitverlauf sichtbar	Der Benutzer will wissen, wann er wieviel gesprochen hat.	Dem Benutzer wird ein Diagramm dargestellt, welches den prozentualen Sprecheranteil in einem bestimmten Zeitabschnitt der Unterhaltung aufzeigt.

Aktualisierung

Dem Benutzer wird ein sich aktualisierendes Diagramm mit den ausgewerteten Daten dargestellt. Der Benutzer kann wählen, in welchen Zeitabständen das Diagramm aktualisiert wird.

Dem Benutzer wird eine Einstellung geboten, in welcher er die Aktualisierungsfrequenz einstellen kann. Der Bereich wird so gewählt, dass er zu keinen technischen Problemen führt und in einem zu erwartenden Bereich liegt.

Tabelle 11 - Faktoren für die Diagramme

Für den Anteil jedes Sprechers über die gesamte Unterhaltung wird ein Kreisdiagramm verwendet. Ein Kreisdiagramm sollte dem Benutzer bereits bekannt sein. Zudem enthält es nicht viele Elemente und Bezeichnungen, weshalb es leicht verständlich sein sollte und trotzdem alle Anforderungen erfüllt.

Für den Verlauf des Gesprächs wird ein gestapeltes Säulendiagramm verwendet. Dabei stellt die X-Achse den Zeitverlauf dar. Die Y-Achse definiert den prozentualen Anteil eines Sprechers.

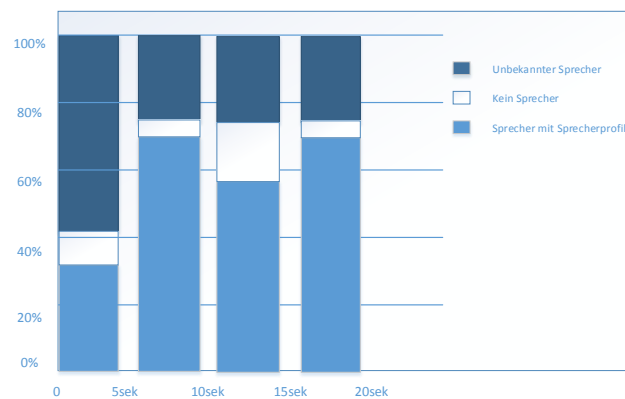


Abbildung 5 - Gestapeltes Säulendiagramm

Dieses Diagramm bietet dem Benutzer einen Zeitverlauf mit derselben Einheit, welche auch beim Kreisdiagramm verwendet wird. Dieser Punkt soll die Lernförderlichkeit des GUIs fördern (siehe Tabelle 9). Als Nachteil wird angenommen, dass nicht in jedem Fall sofort exakt festgestellt werden kann, wieviel Prozent ein Balken darstellt. Dies wird versucht, mit der Diagramm-Bibliothek bestmöglich zu realisieren.

3.1.6 Sprechererkennung

Das Ziel der Applikation ist, anhand eines angelegten Sprecherprofils, eine Unterhaltung zu starten, in welcher mit einem Algorithmus erkannt werden kann, ob der Benutzer spricht. Ebenfalls soll erkannt werden, ob eine Person ohne Benutzerprofil oder niemand spricht. Nehmen mehrere Personen ohne Sprecherprofil an der Unterhaltung teil, so wird zwischen diesen nicht unterschieden. Folglich existieren in einer Unterhaltung die Profile „Niemand“, „Andere“, und jenes des geladenen Sprechermodells.

Abgemacht wurde, dass für das Sprechermodell ein Gaussian-Mixture-Model (siehe Kapitel 2.2.3) eingesetzt wird. Für die Sprechererkennung muss mindestens ein Algorithmus implementiert werden, welcher Audiodaten einliest, für einen Sprechererkennungsalgorithmus aufbereitet und anschliessend ein Modell daraus erstellt, anhand von welchem später ausgewertet werden kann, ob der Sprecher mit dem Sprecherprofil spricht oder nicht. Um die Sprecherbestimmung durchführen zu können, wird ebenfalls eine Komponente benötigt, welche anhand von Kriterien einen Sprecher für die Features bestimmt. Um die Zuweisung an das Profil „Niemand“ durchführen zu können, wird ein Stille-Filter benötigt, welcher die Features auf Stille überprüft.

Kalibrierung

Ist noch kein Benutzerprofil angelegt, was nach jeder Installation der Applikation der Fall ist, so muss eines erstellt werden. In diesem Profil wird die Stimme des Sprechers modelliert. Während der Kalibrierung werden kontinuierlich Audiodaten vom Mikrofon des Handys eingelesen und die Features davon extrahiert. Beendet der Benutzer die Kalibrierung, wird anhand aller extrahierten Features ein Modell erstellt.

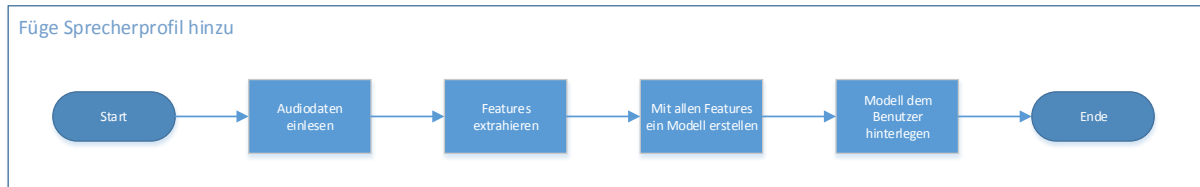


Diagramm 3 - Ablauf Modell generieren

Unterhaltung führen

Ist die Kalibrierung durchgeführt und konnte ein Benutzerprofil mit den eingelesenen Audiodaten erstellt werden, so kann der Sprecher mit seinem Profil in einer Unterhaltung identifiziert werden. Während einer Unterhaltung werden kontinuierlich Audiodaten eingelesen und davon die Features extrahiert. Die Features werden anschliessend einem Profil zugewiesen.

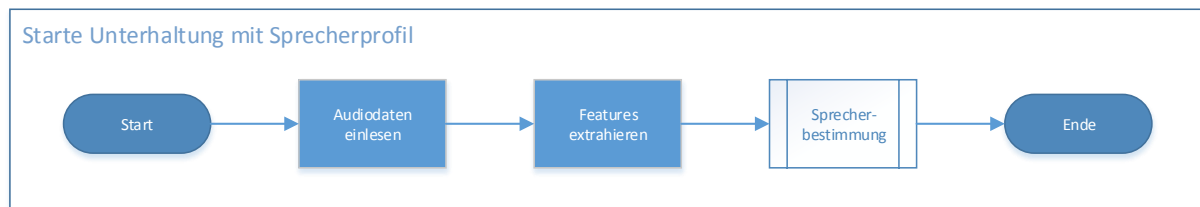


Diagramm 4 - Ablauf Sprechererkennung

Die Bestimmung, zu welchem Profil die Features zugeordnet werden, wird von einer separaten Komponente bestimmt, in dieser Arbeit „Entscheidungslogik für die Sprecherbestimmung“ genannt. Diese Entscheidungslogik muss den Stille-Filter kennen, um ihn überprüfen zu lassen, ob die Features Stille beinhalten. Ebenfalls muss die Komponente das Sprechermodell kennen, um die Wahrscheinlichkeitsberechnung bzw. Log-Likelihood-Methode aufrufen zu können, mit welcher bestimmt wird, ob der resultierende Wert hoch genug ist, um die Features dem Sprechermodell zuzuordnen zu können. Diese Methode wird immer aufgerufen, falls der Stille-Filter die Features nicht als Stille wertet.

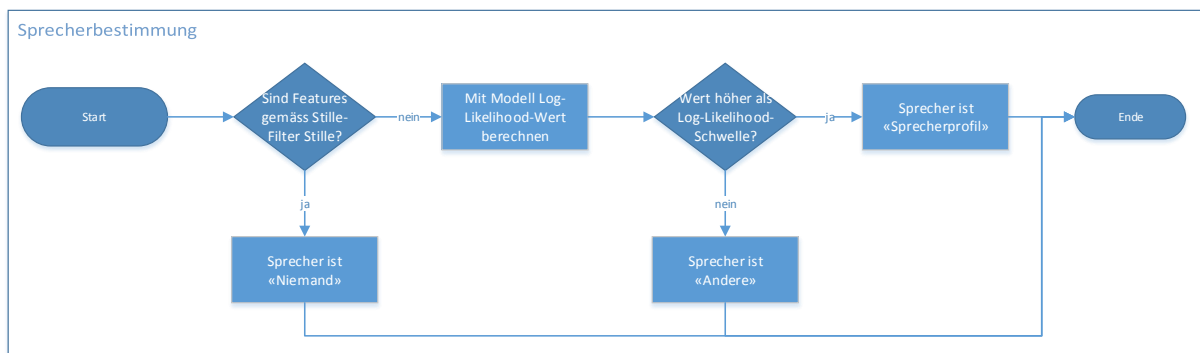


Diagramm 5 - Sprecherbestimmung

Kalibrierung erweitern

Das Erweitern der Kalibrierung hat das Ziel, ein genaueres Sprecherprofil zu bilden. Dafür werden die bisherigen Audiodaten des Sprecherprofils zu den neuen Kalibrierungsdaten hinzugefügt. Daraus entsteht anschliessend ein neues Modell.

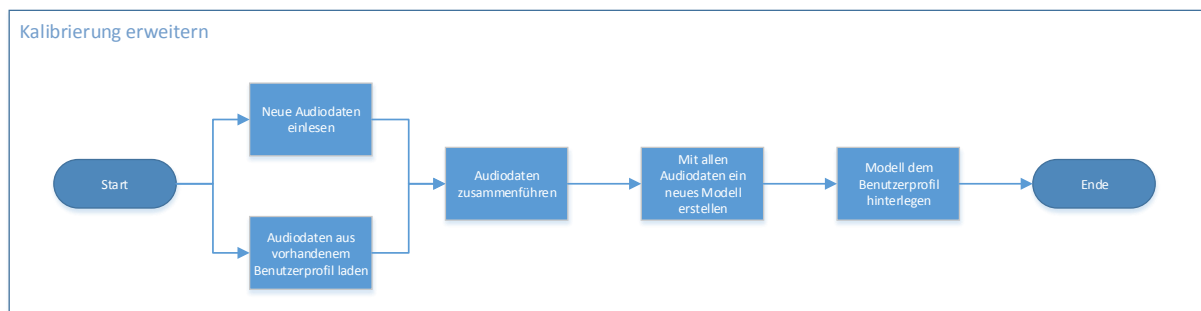


Diagramm 6 - Kalibrierung erweitern

Die Audiodaten des bisherigen Sprechermodells können dabei als WAVE-Datei oder als MFCCs vorliegen, da mit beiden Formaten eine Weiterverarbeitung möglich ist. Es werden jedoch MFCCs für das Speichern der Audiodaten des Benutzerprofils gewählt, weil damit die erneute Konvertierung zu MFCCs entfällt.

3.1.7 Codestruktur

Der Aufbau des Codes muss gemäss Aufgabenstellung so strukturiert werden, dass die einzelnen Komponenten der Software voneinander getrennt sind, so dass sie ohne hohen Aufwand ausgetauscht werden können.

Austauschbare Komponenten

Nachfolgende Komponenten sind entscheidend für die Sprechererkennung. Diese sollen bei Bedarf einfach und störungsfrei ausgetauscht werden können. Es sollen auch mehrere Komponenten ersetzt werden können, ohne dass dies die Ausführbarkeit der Applikation beeinträchtigt.

Komponente	Beschreibung
Parameter ändern oder hinzufügen	Parameter für die Audiodaten-Verarbeitung oder ein Sprechermodell, welche in der gesamten Applikation gleich sind, sollen in zentralen Klassen abgeändert oder neue hinzugefügt werden können.
Features extrahieren	Die eingelesenen Mikrofondaten sind als byte-Array zugänglich. Diese müssen für den Sprechererkennungsalgorithmus passend konvertiert werden.
Filter für Audiodaten hinzufügen oder entfernen	Um nur mit bestimmten eingelesenen Audiodaten oder Bestandteilen davon Berechnungen durchzuführen, etwa nur mit MFCCs welche keine Stille beinhalten, sollen Filter eingesetzt werden können.
Sprechermodell	Das Sprechermodell soll frei implementierbar sein. Dabei soll es keine Rolle spielen, wie die Daten verarbeitet werden. Das Sprechermodell ist in jedem Fall zuständig, ein Modell zu generieren und ein existierendes Modell mit Audiodaten zu vergleichen.
Entscheidungslogik für die Sprecherbestimmung	Um mit der berechneten Wahrscheinlichkeit eines MFCCs zu einem Sprechermodell bestimmen zu können, welcher Sprecher gesprochen hat, können mehrere Faktoren oder Abläufe eine Rolle spielen. Etwa, ob die MFCCs weiter gefiltert werden sollen oder mit welchem Schwellenwert entschieden wird, ob der vom Modell berechnete Log-Likelihood-Wert genug hoch ist, um die MFCCs dem Benutzerprofil zuzuweisen.

Tabelle 12 - Austauschbare Komponenten

3.2 Potentiell relevante Technologien

Um die geeignetste Technologie für die Applikation einsetzen zu können, mussten vorerst vorhandene Technologien überprüft und bewertet werden. Anschliessend wurde bestimmt, welche Technologien eingesetzt werden sollen.

3.2.1 Diagramme

Um die Diagramme zu erstellen, wurden diverse Diagramm-Bibliotheken untersucht. Davon kamen drei in die engere Auswahl. Diese erfüllen viele oder alle Anforderungen. Nachfolgend werden die Diagramm-Bibliotheken mit den Kriterien aufgeführt.

	Androidplot (Version 0.6.0)	AChartEngine (Version 1.1.0)	AFreeChart (Version 0.0.4)
Android als unterstützte Plattform	✓	✓	✓
Dokumentation vorhanden	✓	✓	?
Aktualisieren der Diagramme möglich	✓	✓	✓
Kreisdiagramm vorhanden	✓	✓	✓
Gestapeltes Säulendiagramm vorhanden	✓	✓	✗
Keine Internetkonnektivität notwendig	✓	✓	✓
Frei verfügbar	✓	✓	✓

Abbildung 6 - Bewertung der Diagramm-Bibliotheken

3.2.2 Sprechererkennung

Für die Sprechererkennung müssen grundsätzlich zwei Komponenten vorhanden sein. Eine Komponente, welche aus den Mikrofondaten MFCCs generiert und ein Algorithmus, der das Erstellen eines GMMs und das Testen des Modells mit Features übernimmt. Es wurde deshalb nach einer Bibliothek gesucht, welche diese beiden Komponenten beinhaltet. Gemäss der Android-Developer-Seite [15] wird empfohlen, nicht auf C- oder C++-Code zu setzen, falls dies nicht notwendig ist. Deshalb wird das Hauptaugenmerk auf eine Java-Bibliothek gerichtet.

Zu den beiden Bibliotheken wurde zusätzlich die Option hinzugenommen, den benötigten Code für das Sprechermodell selbst zu implementieren. Dies beschränkt sich jedoch darauf, einen vorhandenen C++-Code [16] in Java zu übersetzen und falls nötig entsprechend anzupassen.

	CoMIRVA (Version 0.36)	PocketSphinx- Android (Version 0.8)	Eigene Implementation
Enthält ein MFCC-Konvertierung	✓	✓	✗
Enthält ein Gaussian-Mixture-Model	✓	✓	✓
Dokumentation	?	✓	?
Benötigt kein NDK	✓	✗	✓
Keine Internetkonnektivität notwendig	✓	✓	✓
Frei verfügbar	✓	✓	✓

Abbildung 7 – Bewertung der Sprechererkennungs-Bibliotheken

3.3 Verwendete Technologien

Nachfolgend werden jene Technologien beschrieben, welche in der Applikation verwendet wurden.

3.3.1 Diagramme

AChartEngine

AChartEngine enthält alle für diese Arbeit nötigen Eigenschaften. Entscheidend ist zudem, dass viele Beispiele existieren, die zeigen, wie vorgegangen werden muss, um Diagramme fortlaufend mit neuen Daten zu versorgen und anschliessend zu aktualisieren.

Grundsätzlich arbeitet AChartEngine mit einer sogenannten „GraphicalView“-Klasse, welche die „View“-Klasse [17] von Android erweitert. So wird bei jedem Zeichnen des Diagramms die „invalidate“-Methode aufgerufen. Diese bewirkt, dass die gesamte View neu gezeichnet wird. Dies kann als Performance-Schwäche angesehen werden. Es wurde jedoch nach eigens durchgeführten Tests keine Probleme damit erkannt.

3.3.2 Sprechererkennung

Für die Sprechererkennung wurden zwei Methoden implementiert. Einerseits hat sich herausgestellt, dass die Geschwindigkeit für das Erstellen eines Sprechermodells bei der CoMIRVA-Bibliothek zu schlecht ist, als in Kapitels 3.1.1 definiert wird. Andererseits ist gemäss Aufgabenstellung das Ziel, eine Architektur zu schaffen, die es erlaubt neue Algorithmen für die Sprechererkennung einfach zu integrieren. Dies kann am besten überprüft werden, wenn die entsprechende Komponente ausgetauscht wird.

CoMIRVA

Die CoMIRVA-Bibliothek präsentiert sich als hervorragende Möglichkeit Sprachdaten zu verarbeiten. Sie bietet Methoden, um aus Mikrofondaten MFCCs zu generieren sowie mit diesen MFCCs und einem ebenfalls enthaltenen k-Means-Algorithmus ein GMM zu erstellen. Dieses kann mit dem vorhandenen EM-Algorithmus trainiert werden. Des Weiteren lässt sich mit der enthaltenen Log-Likelihood-Methode

eine Ähnlichkeit zwischen dem Sprechermodell und untrainierten MFCCs berechnen. Diese Komponenten beschreiben nur einen kleinen Teil der CoMIRVA-Bibliothek. Sie bietet einige weitere Algorithmen für Datenverarbeitung, Audioverarbeitung und Data-Mining.

Die CoMIRVA-Bibliothek wurde bereits im Talkalyzer-Projekt verwendet, weshalb der Ablauf und die benötigten Klassen dort ersichtlich sind. Da der komplette Code in Java geschrieben ist, lässt sich auch das Importieren von NDK vermeiden. Zudem ist für die Bibliothek der Quellcode verfügbar. Dieser war auch nötig, da die Bibliothek nicht für Android entwickelt wurde. So musste eine Klasse angepasst werden.

Im Paket von CoMIRVA wird standardmässig die „AudioPreProcessor“-Klasse verwendet um Audiodaten zu verarbeiten. Diese benutzt jedoch die Klasse „AudioInputStream“ vom „javax“-Paket. Dieses ist für Android nicht verfügbar. Im Internet werden Umgehungslösungen vorgeschlagen. Zur Sicherheit wurde mit dem Entwickler der Klasse, Herrn Klaus Seyerlehner, Kontakt aufgenommen und nachgefragt, welche Lösung er für richtig hält. Gemäss Herrn Klaus Seyerlehner kann anstatt die „AudioInputStream“ auch die „InputStream“-Klasse verwendet werden (siehe Kapitel 8.2.1). Deshalb wurde die „AudioPreProcessor“-Klasse so abgeändert, dass sie mit der „InputStream“-Klasse arbeiten kann bzw. weitere Klassen so angepasst, dass die „AudioPreProcessor“-Klasse nur minimal abgeändert wird.

Für den Sprechererkennungsalgorithmus wurde zuerst die CoMIRVA-Bibliothek verwendet. Es wurde jedoch festgestellt, dass der k-Means-Algorithmus und der EM-Algorithmus nicht performant arbeiten. Es wurden für das Erstellen des Sprechermodells deutlich höhere Zeiten gemessen, als gemäss Kapitel 3.1.1 angestrebt wird. Das Problem für diese Projektarbeit ist, dass bei der CoMIRVA-Bibliothek für die Varianzen zu viel Information gespeichert und verarbeitet wird. Dies hat ein zweidimensionales Array pro Gruppe und ein dreidimensionales Array insgesamt zur Folge. Es liegen somit mehr Daten vor, als gemäss Kapitel 2.2.2 definiert bzw. notwendig ist.

Eigene Implementation

Die eigene Implementation beinhaltet in Java übersetzten C++-Code [16]. Dieser Code enthält das Erstellen des GMMs und das Vergleichen von Features mit dem Modell. Für das Erzeugen von MFCCs wird weiterhin die CoMIRVA-Bibliothek verwendet.

Als Ersatz für den k-Means-Algorithmus der CoMIRVA-Bibliothek wird die Apache Commons Math-Bibliothek eingesetzt. Diese beinhaltet den erweiterten k-Means++-Algorithmus.

3.4 Verworfenе Technologien

Nachfolgend werden jene Technologien beschrieben, welche ebenfalls evaluiert wurden, jedoch für diese Projektarbeit als nicht geeignet resultierten bzw. nicht eingesetzt wurden.

3.4.1 Diagramme

AFreeChart

AFreeChart ist eine auf Android portierte Version von der Bibliothek JFreeChart, welche umfangreiche Möglichkeiten für Java Applets, Servlets und JSPs bietet. Die Aktualisierung der Diagramme wird mit einem Event-Benachrichtigungs-Mechanismus durchgeführt. Für die Vorgehensweise, um ein Diagramm zu aktualisieren, fehlten jedoch Beispiele. Es war nicht klar mit welchen Methoden eine Aktualisierung abläuft. Des Weiteren konnte auch mit der Vorgehensweise von der JFreeChart-Bibliothek keine Aktualisierung erreicht werden. Zudem scheitert AFreeChart an der fehlenden Unterstützung des gestapelten Säulendiagramms, welches gemäss Konzept eingesetzt werden soll. AFreeChart unterstützt dafür jedoch 3D-Diagramme.

Androidplot

Androidplot und AChartEngine bieten einen ähnlichen Funktions- und Dokumentationsumfang. Somit hätte auch Androidplot verwendet werden können. Der Grund, weshalb Androidplot nicht verwendet wurde ist, dass die AChartEngine-Bibliothek in mehr Applikationen verwendet wird, weshalb auf eine grössere Community rückgeschlossen wird und sich eine anhaltendere Software-Unterstützung prognostizieren lässt. Der Prozentanteil für den Einsatz beträgt für Androidplot 0.12% [18] und für AChartEngine 0.7% [19].

3.4.2 Spracherkennung

PocketSphinx Android

PocketSphinx Android gehört zur Gruppe von Spracherkennungssystemen der Carnegie Mellon Universität. Als Gruppenbezeichnung wird CMU Sphinx verwendet. Dazu gehören auch die Versionen Sphinx 4, ein Framework für Spracherkennung in Java geschrieben und PocketSphinx, welche für Embedded Systeme entwickelt wurde.

Die PocketSphinx Android-Bibliothek ist speziell für Android-Geräte entwickelt worden. Sie bietet die benötigten Funktionen, etwa MFCCs zu generieren und GMMs zu erstellen. Ebenfalls ist Dokumentation [20] und Beispiels-Code [21] vorhanden.

Um die Funktionen benutzen zu können, wird ein „Config“-Objekt erstellt. Diesem werden alle Parameter-Keys als String und die Werte vom jeweiligen Datentyp hinzugefügt. Dieser Vorgang ist auch für andere Sphinx-Bibliotheken derselbe. Es werden jeweils dieselben Parameter-Keys für die verschiedenen Bibliotheken verwendet. Anschliessend wird dieses „Config“-Objekt einem „Decoder“-Objekt überwiesen, mit welchem gearbeitet werden kann.

Jedoch enthält die Bibliothek zusätzliche Abhängigkeiten, welche vermieden werden wollen (siehe Kapitel 3.2.2). Dazu gehören Apache Ant [22], SWIG [23] und Android NDK [15]. Es ist zudem anzumerken, dass die Bibliothek für Spracherkennung ausgerichtet ist, nicht für Sprecherkennung. Um für eine ähnliche Arbeit die Features zu generieren, wird sie jedoch erfolgreich eingesetzt [24].

4 Resultat

Nachfolgend werden die erhaltenen Resultate beschrieben. Es wird beschrieben, wie das GUI, die Codestruktur und die Sprechererkennung implementiert sind.

4.1 GUI

Für das GUI werden wenn möglich Standardkomponenten, welche Android zur Verfügung stellt, verwendet. Zwei verwendete Icons wurden von der „flaticon.com“-Webseite bezogen. Dieser Umstand wird im Kapitel 6.2.2 thematisiert. Die Bildschirmfotos sind mit einem „Nexus 4“-Gerät [25] erstellt worden. Insgesamt werden fünf Activities eingesetzt, um das GUI zu verwalten.

Activities	
Launch-Activity	Ist für die Startseite der Applikation zuständig. Diese wird in jedem Fall gestartet, bleibt bei einer Kalibrierung jedoch im Hintergrund.
Calibration-Activity	Ist für die Kalibrierung zuständig.
Talk-Activity	Ist für eine Unterhaltung zuständig.
TalkAnalysis-Activity	Ist für die detailliertere Analyse einer Unterhaltung zuständig.
ExtendCalibration-Activity	Ist für die Erweiterung der Kalibrierung zuständig.

Tabelle 13 - Implementierte Activities

Die „Calibration“-Activity und die „ExtendCalibration“-Activity sind sich ziemlich ähnlich. Für eine bessere Codestruktur wurden die beiden Activities getrennt, erweitern jedoch die abstrakte „AbstractCalibration“-Activity, welche die Gemeinsamkeiten enthält.

4.1.1 Kalibrierung

Startet der Benutzer die Applikation das erste Mal oder es ist kein Benutzerprofil vorhanden, so wird die Kalibrierung bzw. die „Calibration“-Activity gestartet. Er wird anfangs aufgefordert seinen Namen einzugeben und Einleitungstext zu lesen, um zu verstehen, weshalb die Kalibrierung notwendig ist und wie er dabei vorgehen muss.

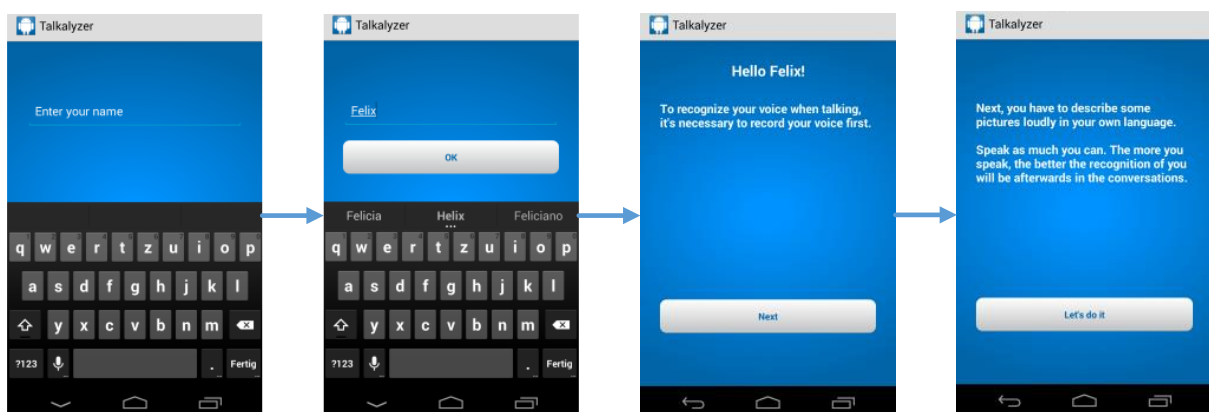


Abbildung 8 – Vorarbeit für Kalibrierung in der Calibration-Activity

Anschließend werden ihm Bilder präsentiert, welche er laut und in seiner Sprache beschreiben muss (siehe Kapitel 3.1.5). Es werden Bilder von der Webseite „<http://morguefile.com/>“ verwendet. Diese Bilder sind frei verfügbar und dürfen frei weiterverarbeitet und verwendet werden [26]. Falls der Be-

nutzer ein anderes Bild beschreiben möchte, so kann über eine Schaltfläche zum nächsten Bild gewechselt werden. Ebenfalls besteht die Möglichkeit die Kalibrierung zu pausieren und später damit fortzuführen.

Dem Benutzer wird stets mit einem Fortschrittsbalken angezeigt, wieviel Daten bereits eingelesen wurden und wie viel dies von der erwarteten Menge ausmacht. Dieser Fortschrittsbalken startet auf der linken Hälfte mit der Farbe Rot, um zu signalisieren, dass noch zu wenig gesprochen wurde und endet auf der rechten Seite mit der Farbe Grün, womit angezeigt wird, dass viel gesprochen wurde. Die erwartete Menge an Daten kann vom Applikationsentwickler über einen Parameter eingestellt werden. Entsprechend verhält sich der Fortschrittsbalken.

Ebenfalls wird dem Benutzer mittels eines langsam blinkenden Texts signalisiert, dass vom Handy aufgenommen wird. Werden die Audiodaten etwa nach Lautstärke gefiltert und es werden während vier Sekunden keine neuen Audiodaten zur Verarbeitung übergeben, so wird dem Benutzer eine Meldung ausgegeben, dass zu viel Hintergrundgeräusche existieren. Dies ist nur eine Annahme, macht dem Benutzer jedoch klar, dass etwas mit der Lautstärke nicht in Ordnung ist.

Wird die „Calibration“-Activity verlassen, etwa weil eine andere Applikation aufgerufen wird, so wird die Kalibrierung automatisch pausiert. Dies wurde umgesetzt, da angenommen wird, dass der Benutzer sich einer anderen Tätigkeit widmet und die nachfolgenden Audiodaten nicht mehr gewinnbringend für die Kalibrierung sind. Kehrt der Benutzer zur „Calibration“-Activity zurück, hat er die Möglichkeit, die Kalibrierung mit den bisherigen Daten fortzusetzen. Verlässt er die Kalibrierung ohne ein Modell zu erstellen, wird er beim nächsten Start der Applikation wieder mit der Kalibrierung beginnen müssen.

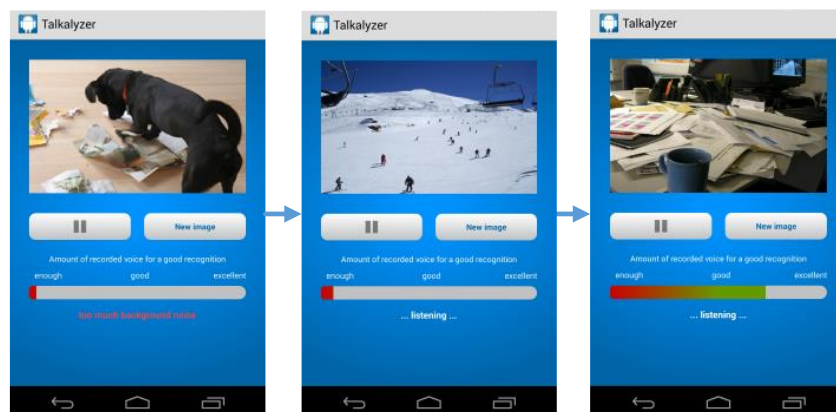


Abbildung 9 - Bilder beschreiben in der Calibration-Activity

Nach dem Erreichen der minimal erwarteten Menge an Daten ist es dem Benutzer erstmals möglich, die Kalibrierung so zu beenden, dass sein Modell berechnet werden kann. Andernfalls wird ihm mitgeteilt, dass er noch weiter sprechen muss. Ebenfalls existiert keine Schaltfläche, um ein Modell zu berechnen.

Wählt der Benutzer die Schaltfläche an, um die Kalibrierung zu beenden, so muss er noch warten bis sein Modell berechnet wurde. Genauere Details zur Berechnungszeit können im Kapitel 0 nachgelesen werden. Während der Wartezeit wird dem Benutzer nochmals Text angezeigt, etwa dass er sein Sprechermodell für bessere Resultate erweitern kann. Dies ist kein besonders wichtiger Text, weshalb es auch nicht als schlimm gewertet wird, falls der Benutzer den Text nicht liest, etwa weil das Modell sofort berechnet wurde. Zudem wird der Einsatz einer Hilfsseite im Kapitel 6.2.2 thematisiert.

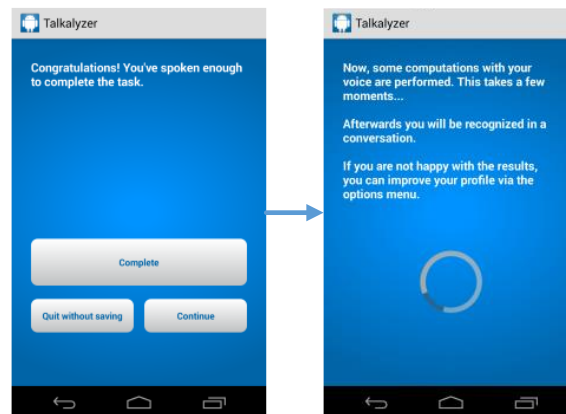


Abbildung 10 – Kalibrierung abschliessen in der Calibration-Activity

Schlägt die Kalibrierung fehl bzw. kein Modell konnte erstellt werden, ausgelöst etwa durch eine schlechte Wahl des Startzentrums vom k-Means-Algorithmus oder die in der Applikation definierten Parameter, so wird dem Benutzer per Schaltfläche ermöglicht, mit den Daten nochmals ein Modell zu erstellen. Ein Fehlschlagen des Algorithmus wird als schlecht eingestuft, weshalb hiermit versucht wird, den Benutzer nochmals zu motivieren, die Kalibrierung abzuschliessen und die Applikation anschliessend zu gebrauchen. Es wird jedoch erwartet, dass dies äusserst selten vorkommt, da dies in der Testphase vereinzelt und nur mit Daten von weniger als zehn Sekunden auftrat.

4.1.2 Einstiegsseite

Wurde mit den Daten der Kalibrierung ein Sprecherprofil erstellt und dem Profil hinterlegt, so wird dieser Benutzer als Standardbenutzer in der Applikation definiert. Er erreicht anschliessend die Einstiegsseite, welche von nun an bei jedem Start der Applikation gezeigt wird. Diese wird von der „Launch“-Activity verwaltet. Dem Benutzer wird eine Willkommensmeldung mit seinem Profilnamen angezeigt. Ebenfalls ist eine Schaltfläche vorhanden, um eine Unterhaltung zu starten. Diese Bildschirmseite ist sehr leer. Gedacht war, dass hier das Applikations-Icon platziert wird, entweder als Schaltfläche um eine Unterhaltung zu starten oder als Hintergrundbild. Das Icon, welches in Auftrag gegeben wurde, konnte jedoch nicht erhalten werden. Ein Benutzer besitzt hier ebenfalls die Möglichkeit in die Optionen zu wechseln. Dort kann er ein neues Benutzerprofil anlegen, ein Benutzerprofil löschen, das Standardbenutzerprofil festlegen oder die Kalibrierung seines Profils erweitern. Die Möglichkeit, ein weiteres Profil zu erstellen, wäre nicht nötig, da die Applikation nur für ein Profil vorgesehen ist und falls der Benutzer sein Profil löscht, muss er in jedem Fall wieder ein neues Profil erstellen. Trotzdem wird diese Funktion implementiert gelassen, etwa wegen der Evaluierung und für weitere Tests. Ebenfalls ist ein Icon mit einem Fragezeichen ersichtlich. Dort war vorgesehen gewesen, eine Hilfsseite einzubinden (siehe Kapitel 6.2.2).

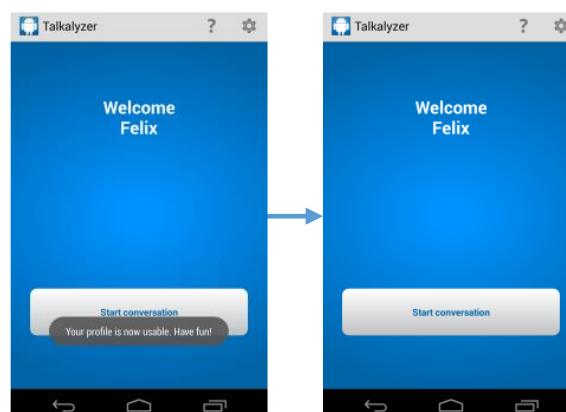


Abbildung 11 – Launch-Activity

4.1.3 Unterhaltung führen

Wählt der Benutzer auf der Startseite die Schaltfläche um eine neue Unterhaltung zu starten an, so wird die Unterhaltung bzw. die „Talk“-Activity gestartet. Insgesamt wird eine Bildschirmseite für die Unterhaltung verwendet. Während einer Unterhaltung soll der Benutzer auf dem aktuellen Stand betreffend Sprecheranteile gehalten werden. Dafür existiert ein Kreisdiagramm, welches jeweils die neuesten aufbereiteten Daten anzeigt. Zusätzlich wird dem Benutzer eine Stoppuhr angezeigt, damit er weiss, wie lange die Unterhaltung von der Applikation bereits analysiert wird. Ebenfalls wird dem Benutzer mit einem langsam blinkenden, roten Kreis rechts oben signalisiert, dass die Unterhaltung analysiert wird.

Pausiert der Benutzer die Unterhaltung, so werden alle unverarbeiteten Daten zu Ende verarbeitet. Dies hängt mit dem Parameter zusammen, welcher definiert, wie viele MFCCs die Sprecherbestimmung pro Verarbeitungsschritt beinhalten soll. Dieser Zwischenschritt dauert weniger als eine Sekunde. Dass noch Daten verarbeitet werden, wird dem Benutzer mit einem kleinen Fortschrittsbalken angezeigt. Wählt er während dieser Zeit eine Schaltfläche an, so erscheint die Meldung bzw. ein Toast [27], dass noch Arbeiten von der Applikation abgeschlossen werden. Es erscheint zudem ein weiterer Button, mit welchem ermöglicht wird, detailliertere Informationen zu den Sprecheranteilen der Unterhaltung zu erhalten.

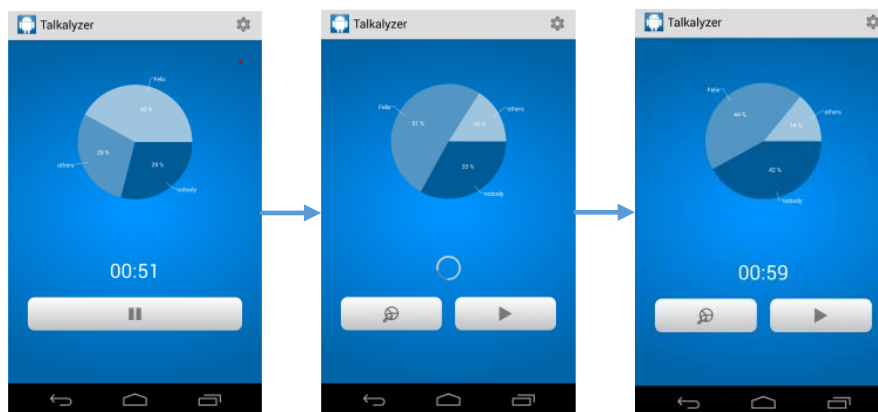


Abbildung 12 – Talk-Activity

Dem Benutzer ist es möglich, die Talkalyzer-Applikation zu verlassen und eine andere Applikation zu starten, ohne dass diese aufhört, die Daten aufzubereiten. Will der Benutzer, dass das Diagramm weniger oder häufiger aktualisiert wird, so kann er dies im Menü Optionen der „Talk“-Activity ändern. Ihm steht eine Auswahl an vordefinierten Werten zur Verfügung zwischen welchen er auswählen kann. Hier wurden bewusst vordefinierte Werte genommen, da dies die Einfachheit des Programms positiv beeinflusst.

Wählt er diese Schaltfläche an, so wird die „TalkAnalysis“-Activity geladen. Dort wird dem Benutzer wieder ein Kreisdiagramm angezeigt. Weiter ist es dem Benutzer möglich, den Zeitbereich anzupassen, in welchem die Daten angezeigt werden sollen. Die Daten des Kreisdiagramms werden jeweils dem ausgewählten Zeitbereich angepasst. Dies wurde mit einer erweiterten SeekBar [28], einer sogenannten RangeSeekBar erreicht. Diese besteht aus einem Slider und zwei Thumbnails [29]. Sie erlaubt es, zwei Werte auf einem Auswahlbereich festzulegen anstatt nur einen.



Abbildung 13 – RangeSeekBar

Die RangeSeekBar ist standardmässig nicht in Android verfügbar. Code für eine RangeSeekBar ist jedoch im Internet zu finden [29]. Dieser wurde auch verwendet. Zusätzlich wurde zur RangeSeekBar implementiert, dass die Zeit des aktuellen Bereichs mit Labels angezeigt wird. Anfangs wurde implementiert, dass die Labels unter den beiden Thumbnails stehen. Je nach ausgewähltem Bereich müssen die Labels jedoch entfernt vom dazugehörigen Thumbnail platziert werden. Dies kann den Benutzer verwirren. Deshalb wurde implementiert, dass die Start- und Endzeit jeweils am linken und rechten Bildschirmrand angezeigt. Der Code für die dynamisch anpassbaren Labels ist jedoch noch vorhanden und kann verwendet werden.

Auf einer weiteren Bildschirmseite erhält der Benutzer die Möglichkeit, die Sprecheranteile als Zeitverlauf zu erhalten. Dazu wurde ein gestapeltes Säulendiagramm erstellt. Eine Säule stellt dabei einen Zeitbereich dar. Dem Benutzer wird ermöglicht, mit Wischen und Zoomen sich auf der Zeitachse des Diagramms zu bewegen. Eine genauere Beschreibung dazu ist im Kapitel 4.1.5 ersichtlich.



Abbildung 14 - TalkAnalysis-Activity

4.1.4 Kalibrierung erweitern

Um die Kalibrierung bei einem bestehenden Profil zu erweitern, kann der Benutzer dies über die Optionen beim Startbildschirm auswählen. Anschliessend wird die „ExtendCalibration“-Activity geladen. Der Benutzer wird wie bei der Kalibrierung aufgefordert, Bilder zu beschreiben. Die Namenseingabe und der Einleitungstext werden jedoch weggelassen. Ist der Benutzer fertig mit Bilder Beschreiben, so hat er die Möglichkeit, ein neues Modell zu berechnen oder die neu kalibrierten Daten zu verwerfen, etwa aus dem Grund, falls der Benutzer während dem Beschreiben abgelenkt wurde. So wird verhindert, dass sein bisheriges Modell ebenfalls unbrauchbar wird. Das Verhalten der „Calibration“- und der „ExtendCalibration“-Activity ist dasselbe.

4.1.5 Diagramme

Für die Applikation wurden drei Diagramme mit der AChartEngine-Bibliothek umgesetzt. Ein Kreisdiagramm zeigt dem Benutzer während einer Unterhaltung die aktuell ausgewerteten Daten an. Das Diagramm enthält die Sprecheranteile für das Benutzerprofil sowie den Anteilen für die Profile „Andere“ und „Niemand“ (siehe Kapitel 3.1.6) in Prozent dargestellt. Das Diagramm wird in einem bestimmten Intervall aktualisiert. Standardmässig beträgt die Aktualisierungszeit zehn Sekunden. Dies kann jedoch vom Benutzer in den Optionen geändert werden.

Beim gestapelten Säulendiagramm werden die Säulen mit dem Parameter „STACKED“ hintereinander platziert, anstatt wie gewünscht aufeinander. Es musste also noch zusätzlicher Code geschrieben werden, welcher dies korrigiert. Es ist eine inoffizielle Version der AChartEngine vorhanden, die diesen Fehler korrigieren soll [30], indem man den Balken-Diagrammtyp von „STACKED“ auf „HEAP“ ändert. Diese Version wurde jedoch nicht verwendet, da sie nicht auf der offiziellen Seite von AChartEngine verfügbar ist. Der selbst implementierte Code wurde auch nicht mit dieser inoffiziellen Version geschrieben. Die Labels für die einzelnen Säulen werden ebenfalls mit eigenem Code erzeugt.

Beim gestapelten Säulendiagramm existieren Fehler, welche als störend empfunden werden können. Um dem Benutzer zu ermöglichen, sich auf der Zeitachse schnell fortzubewegen, wurde das Zoomen ermöglicht. Falls man sich auf der Zeitachse beim Startpunkt befindet und herausgezoomt wird, ist es nicht mehr möglich, sich auf der X-Achse zu bewegen. Teilweise tritt dies auch sonst auf.

Eine Lösung wäre, das gestapelte Säulendiagramm ebenfalls an die RangeSeekBar einzubinden, so dass nur ein bestimmter Zeitabschnitt angezeigt wird. Dies wurde implementiert. Dafür hat das Fragment [31] bzw. die Bildschirmseite mit dem gestapelten Säulendiagramm die selbsterstellte Schnittstelle

„IRangeBarAttachedChartFragment“ implementiert und wurde zum RangeSeekBar-Listener hinzugefügt. Die Implementation funktioniert und wäre verfügbar. Jedoch ist das Zeichnen des Diagramms fehlerhaft. Wird ein kleinerer Zeitbereich ausgewählt, so werden die Säulen mit den korrekten Werten gezeichnet, jedoch bleiben die nachfolgenden Säulen, welche nicht mehr zum ausgewählten Bereich gehören, bestehen. Dieses Problem wurde aus zeitlichen Gründen nicht weiter verfolgt.

Wird im gestapelten Säulendiagramm nur eine Säule gezeichnet, etwa weil der Benutzer direkt in die „TalkAnalysis“-Activity wechselt oder für das Aktualisierungsintervall einen grossen Wert gewählt hat, so wird diese eine Säule mit falschen Proportionen gezeichnet.

Ebenfalls kann die Farbe der Labels von der Legende nicht festgelegt werden. Es werden in jedem Fall diejenigen Farbe genommen, welche auch für die Säulen verwendet werden. Dies führt bis zur Unleserlichkeit. Behoben werden könnte dies durch die Funktion „setMarginsColor“ von der AChartEngine-Klasse „XYMultipleSeriesRenderer“. Diese färbt jedoch nicht nur den Hintergrund der Legende, sondern den gesamten Bereich ausserhalb des Diagramms. Dies wird als keine Verbesserung wahrgenommen. Ebenfalls können die Achsen-Beschriftungen und die Legende nicht frei platziert werden.

Des Weiteren können die Labels auf der Zeitachse nicht wunschgemäss platziert werden. Diese sollten am Ende der jeweiligen Säule stehen, so dass für den Benutzer besser ersichtlich ist, dass es sich bei einer Säule um einen Zeitbereich handelt. Es kann festgelegt werden, ob die Labels links, rechts oder in der Mitte stehen sollen. Dies behebt dieses Problem jedoch nicht, da kein wesentlicher Unterschied entsteht. Es existiert eine Methode, um das Padding für die X-Labels festzulegen. Diese definiert jedoch nur das Padding zwischen der X-Achse und den Labels.

4.2 Codestruktur

Das Ziel dieser Arbeit ist ebenfalls, einfach einzelne Komponenten in der Applikation austauschen zu können. Ebenfalls soll es möglich sein, mehrere Komponenten davon auszutauschen, ohne dass die Funktionalität des Programms beeinflusst wird. Die austauschbaren Komponenten enthalten in den nachfolgenden Abbildungen und Diagrammen weissen Hintergrund. Die nachfolgende Abbildung zeigt, die Beziehungen der austauschbaren Komponenten auf. Diese soll als Überblick dienen.

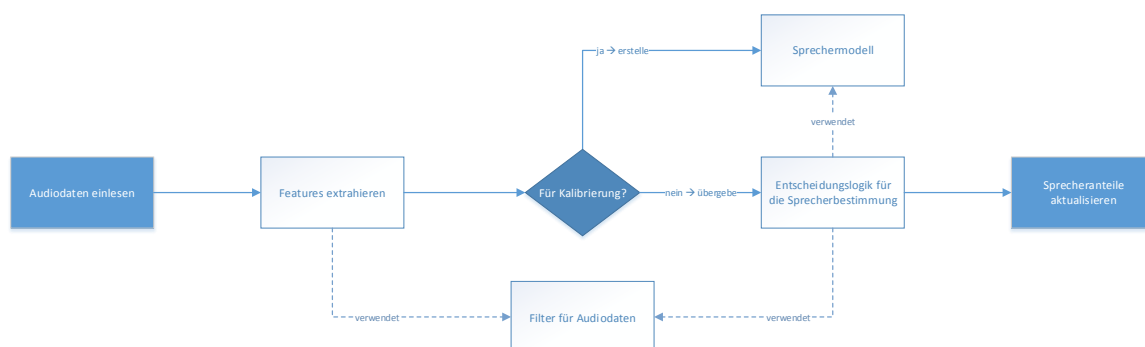


Diagramm 7 – Übersicht austauschbare Komponenten

Bei den austauschbaren Komponenten werden Schnittstellen eingesetzt, um die Methoden festzulegen, welche bei jeder eingesetzten Komponente vorhanden sein müssen. Schnittstellen enthalten am Anfang ihres Namens jeweils den Buchstaben „I“. Jene Klassen, welche mit Features arbeiten, sind gemäss Absprache auf zweidimensionale double-Arrays ausgerichtet. Nachfolgend werden diejenigen Klassen und Schnittstellen erläutert, welche für den Sprechererkennungsalgorithmus relevant sind. Es wird jeweils zuerst die Klasse oder die Schnittstelle mit Text erklärt. Anschliessend wird eine Übersicht derjenigen Klassen in einem UML-Diagramm dargestellt. Wird keine Multiplizität angegeben, so wird jeweils die Multiplizität „1“ verwendet.

Nachfolgend werden jene Klassen und Schnittstellen erklärt welche für die Kalibrierung sowie für eine Unterhaltung verwendet werden und deshalb gleich sind. Dies trifft für das Einlesen von Audiodaten

und deren Konvertierung zu Features zu. Grundsätzlich ist der Ablauf dafür, dass vom Mikrofon des Handys die Daten eingelesen, davon die Features erzeugt und in einem Puffer für die Weiterverarbeitung zwischengespeichert werden.

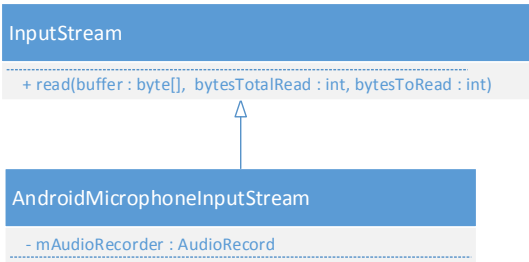
Klasse oder Schnittstelle	Beschreibung
IFeatureConverter	Diese Schnittstelle enthält eine Methode, mit welcher Features angefordert werden. 
Diagramm 8 - UML-Diagramm für die Schnittstelle IFeatureConverter	
AudioStream-ConverterRunnable	Mit diesem Runnable und den darin enthaltenen Klassen werden die Audiodaten in einem Thread kontinuierlich eingelesen, verarbeitet, weitergegeben und zwischengespeichert.
AndroidMicrophoneInputStream	Mit der Methode „read“ dieser Klasse können die Mikrofondaten des Handys eingelesen werden. Die Instanz wird in der Klasse „AudioStream-ConverterRunnable“ erzeugt, so dass beim Beenden des Runnables das Mikrofon, wie in Android vorgesehen, freigegeben werden kann [32] und der Klasse, welche die Schnittstelle „IFeatureConverter“ implementiert mitgegeben. 
Diagramm 9 - UML-Diagramm für die Klasse AndroidMicrophoneInputStream	

Tabelle 14 - Beschreibung der Klasse AudioStreamConverterRunnable und AndroidMicrophoneInputStream sowie der Schnittstelle IFeatureConverter

In dieser Arbeit werden die Audiodaten zu MFCCs konvertiert. Dies geschieht mit Klassen der CoMIRVA-Bibliothek, weshalb für die Komponente für die Extraktion der Features die Klasse „MfccConverterComirva“ implementiert wurde. Diese implementiert die Schnittstelle „IFeatureConverter“. Ebenfalls wird dieser Klasse eine Instanz der Klasse „AndroidMicrophoneInputStream“ mitgegeben, mit welcher die Mikrofondaten eingelesen werden können. Die Klasse bestimmt nicht, wann Daten eingelesen werden. Sie liest und konvertiert Audiodaten jeweils, wenn über die Methode „getNextFeatures“ (siehe Diagramm 8) Features angefordert werden.

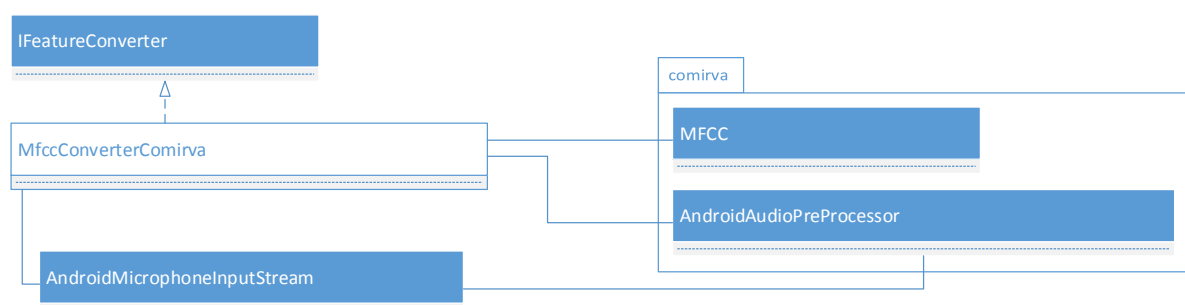


Diagramm 10 - UML-Diagramm der implementierten Feature Extraktion

Die von der „MfccConverterComirva“-Klasse erhaltenen MFCCs werden anschliessend in einer Klasse gespeichert, welche die „IConvertedAudioData“-Schnittstelle implementiert. Die implementierte Klasse mit dieser Schnittstelle heisst „MfccData“.

Klasse oder Schnittstelle	Beschreibung
IConvertedAudioData	Die „MfccData“-Klasse implementiert die „IConvertedAudioData“-Schnittstelle und wird als Puffer für die konvertierten Audiodaten verwendet, bevor sie weiterverarbeitet werden. Deshalb existiert eine Methode für das Hinzufügen und Holen von Features. Ebenfalls können die eingesetzten Filter abgerufen und in weiteren Komponenten der Applikation eingesetzt werden, etwa von der Komponente für die Entscheidungslogik der Sprecherbestimmung. <pre> <<Interface>> IConvertedAudioData + getFeatures(int numberOfFeatures, int stepSize, boolean wantExactNumber) : double[][] + addFeatures(features : double[][], useForCalibration : boolean) + public getFeaturesWithoutRemoving(numberOfFeatures : int) : double[][] + getSilenceFilter() : IFilterSilence </pre>

Diagramm 11 - UML-Diagramm für die Schnittstelle IConvertedAudioData

MfccData	Im „MfccData“-Objekt werden die Features solange zwischengespeichert, bis sie von einem Objekt angefordert werden, etwa für die Modellerzeugung oder zum Testen mit einem Modell. Sollen die Features für die Kalibrierung gefiltert werden, so wird dies mit dem in dieser Klasse angegebenen Filter durchgeführt. Sie ist nicht für den Austausch bestimmt, bildet jedoch eine Kernkomponente der Applikation, weshalb sie erwähnt wird. <pre> classDiagram class IConvertedAudioData { <<Interface>> } class MfccData MfccData -- > IConvertedAudioData </pre>
-----------------	---

Diagramm 12 - UML-Diagramm für die Klasse MfccData

MfccSilenceFilter und IFilterSilence	Das „MfccSilenceFilter“-Objekt wird verwendet, um MFCCs nach Lautstärke zu filtern und zu bestimmen, ob eine Menge von MFCCs als Stille gewertet wird. <pre> classDiagram class MfccData class MfccSilenceFilter class IFeatureFilter { <<Interface>> } class IFilterSilence { <<Interface>> } MfccData -- MfccSilenceFilter MfccSilenceFilter ..> IFilterSilence IFilterSilence -- > IFeatureFilter </pre>
---	--

Diagramm 13 - UML-Diagramm der Klasse MfccSilenceFilter und der Schnittstelle IFilterSilence

Tabelle 15 - Beschreibung der Klassen MfccData und MfccSilenceFilter sowie den Schnittstellen IConvertedAudioData und IFilterSilence

Die Übersicht für die gesamte Konvertierung der Audiodaten ist nachfolgend dargestellt. Das Objekt für die „mStateObject“-Variable ist für die Kalibrierung das „Calibration“- (siehe Kapitel 4.2.1) und für die Unterhaltung das „Talk“-Objekt (siehe Kapitel 4.2.2).

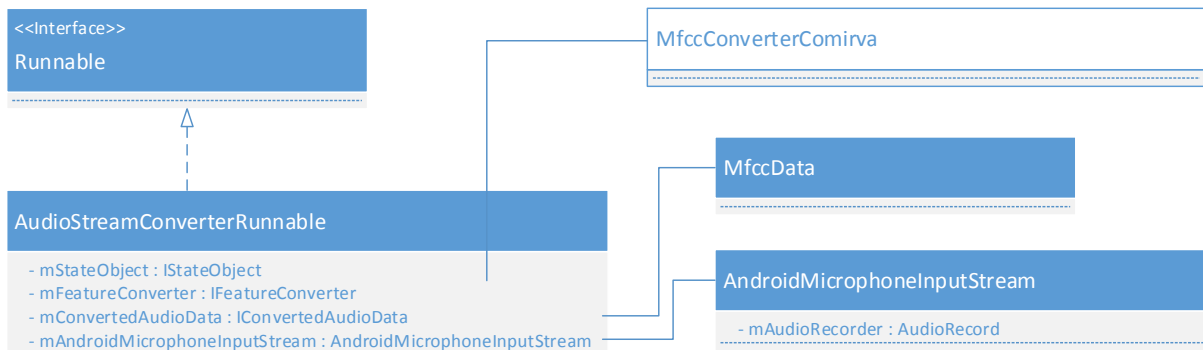


Diagramm 14 - UML-Diagramm Audiodaten einlesen und verarbeiten

Weil für die Sprechererkennung mehrere Threads eingesetzt werden (siehe Kapitel 0), welche kontinuierlich Daten verarbeiten, wird eine Bedingung benötigt, welche die Laufzeit dieser Threads bestimmt.

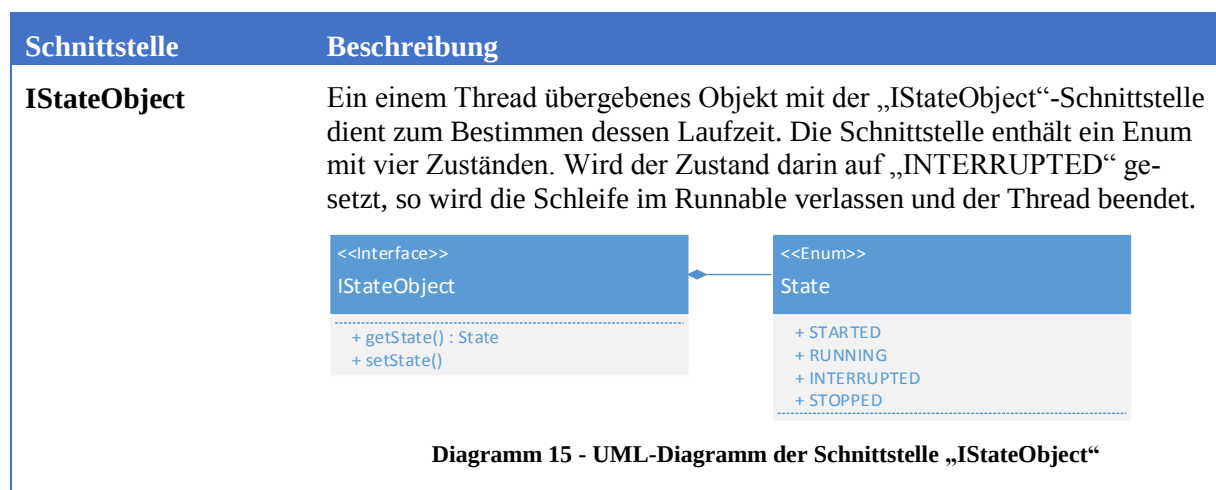


Tabelle 16 - Beschreibung der Schnittstelle IStateObject

Ebenfalls ist für die Kalibrierung sowie für eine Unterhaltung ein Sprechermodell relevant.

Klasse und Schnittstelle	Beschreibung
GaussianMixture-Model und IDefaultTalker-Model	Die Schnittstelle „IDefaultTalkerModel“ enthält eine Methode für die Kalibrierung, um ein Modell mit den eingelesenen Audiodaten bzw. MFCCs erstellen zu können. Diese Methode wird für die erstmalige Kalibrierung sowie für das Erweitern der Kalibrierung verwendet. Ebenfalls ist eine zweite Methode definiert, welche ein Resultat für die Berechnung der Wahrscheinlichkeit von übergebenen MFCCs zum Sprechermodell zurückgibt. Dieser Rückgabewert ist als double-Array definiert, um die Wahrscheinlichkeit von jeder Dimension der Features zusätzlich zur gesamten Wahrscheinlichkeit zurückgeben zu können. Des Weiteren wird eine „set“- und eine „get“-Methode für die MFCCs erzwungen, um die Erweiterung der Kalibrierung realisieren zu

können und das Speichern und Laden der MFCCs benutzerfreundlich durchführen zu können. Denn nicht jedes Mal werden die MFCCs von der Applikation benötigt, wenn das Sprechermodell geladen wird. Deshalb kann dadurch die Bedienbarkeit für den Benutzer gesteigert werden, da die MFCCs nicht von der Festplatte in den Speicher des Handys geladen werden müssen. Die MFCCs werden daher getrennt vom Modell abgespeichert.

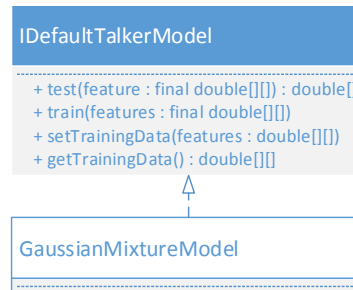


Diagramm 16 - UML-Diagramm der Klasse GaussianMixtureModel und der Schnittstelle IDefaultTalkerModel

Tabelle 17 - Beschreibung der Klasse GaussianMixtureModel und der Schnittstelle IDefaultTalkerModel

4.2.1 Kalibrierung

Das Ziel der Kalibrierung ist, ein Sprechermodell von einer Stimme zu erstellen. Hierfür kommen drei Klassen dazu, welche nicht ersetzt werden können. Dieses Kapitel ist nicht relevant für die austauschbaren Komponenten. Für das Verständnis der gesamten Applikation und des gesamten Dokuments jedoch schon.

Klasse	Beschreibung
Calibration	Das „Calibration“-Objekt stellt die Verbindung zwischen einem neuen Sprecherprofil und seinen Audiodaten dar, bevor das Sprecherprofil erstellt wurde. Es implementiert die Schnittstelle „IStateObject“ und definiert somit die Laufzeit eines Threads. <pre> classDiagram class Calibration { -mTalkerName : String } class IStateObject { <<Interface>> +getState() : State +setState() } Calibration .. > IStateObject </pre> Diagramm 17 - UML-Diagramm der Klasse Calibration Die Instanz wird von der „Calibration“-Activity und der „ExtendCalibration“-Activity erzeugt.
Model-Creator-Runnable	Das „ModelCreatorRunnable“-Runnable erstellt durch das Ausführen der „train“-Methode, welche die Klasse des Sprechermodells wegen der „IDefaultTalkerModel“-Schnittstelle zur Verfügung stellt, das Sprechermodell. Die dafür benötigten MFCCs werden im „MfccData“-Objekt, welches ihm im Konstruktor mitgegeben wird, geholt. Das Runnable wird für die Kalibrierung sowie für das Erweitern der Kalibrierung verwendet. Es benötigt kein „IStateObject“-Objekt, weil das Erzeugen des Sprechermodells aus der Sicht des Runnables „ModelCreatorRunnable“ kein iterativer Prozess ist. Die Speicherung ist nicht enthalten, weil das Modell nicht in

jedem Fall gespeichert werden soll. Es wird dafür eine Methode definiert, mit welcher das aktuell berechnete Modell abgerufen werden kann.

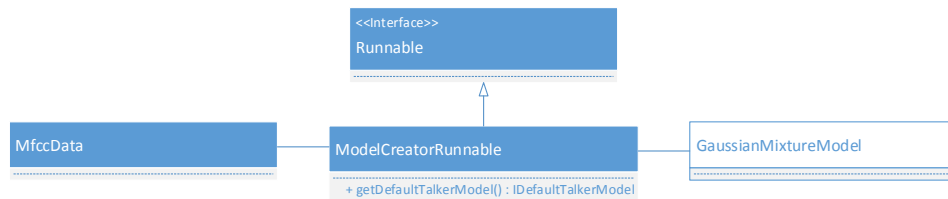


Diagramm 18 - UML Diagramm für die Klasse ModelCreatorRunnable

Calibration-Service

Der Kalibrierungs-Service übernimmt die rechenintensiven Aufgaben der „Calibration“-Activity, etwa das Einlesen und Verarbeiten von Audiodaten sowie das Erstellen des Sprecherprofils. Er führt zwei Threads nacheinander aus. Zuerst werden die Audiodaten mit dem Runnable „AudioStreamConverterRunnable“ eingelesen und zu MFCCs verarbeitet. Beendet der Benutzer die Kalibrierung, so werden alle aufbereiteten MFCCs dem „ModelCreatorRunnable“-Runnable zum Trainieren übergeben.

Tabelle 18 -Beschreibung der Klassen Calibration, ModelCreatorRunnable und CalibrationService

In nachfolgendem UML-Diagramm sind die Beziehungen der Klassen für die Kalibrierung zusammengefasst. Das „AudioStreamConverterRunnable“-Objekt stellt das Objekt gemäss Diagramm 14 dar, ergänzt durch das „IStateObject“-Objekt „Calibration“.

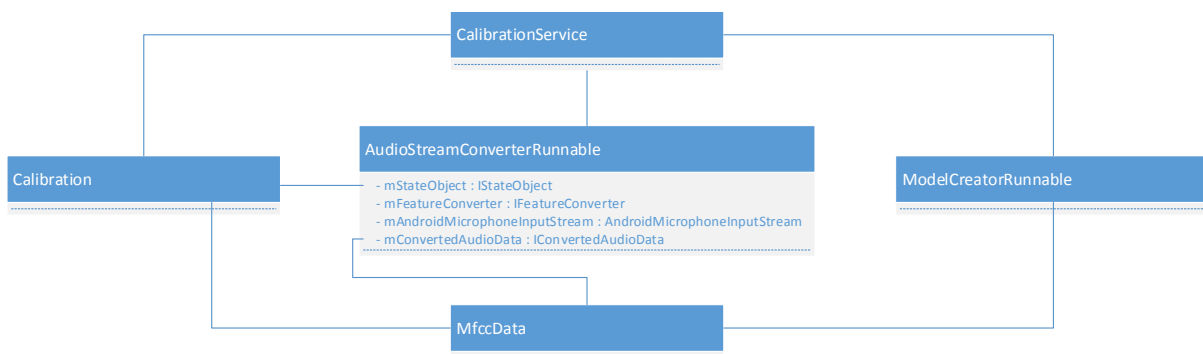


Diagramm 19 - UML-Diagramm Kalibrierung

4.2.2 Unterhaltung führen

Für die Unterhaltung wird fortlaufend entschieden, welcher Sprecher bei den eingelesenen Audiodaten jeweils gesprochen hat. Dafür wird eine Entscheidungslogik benötigt, welche die „IDecisionMaker“-Schnittstelle verwendet.

Klasse und Schnittstelle	Beschreibung
DecisionMaker und IDecisionMaker	Die „IDecisionMaker“-Schnittstelle definiert, dass für eine Anzahl Features denjenigen Sprecher zurückgegeben wird, welcher gemäss Entscheidungslogik für diese Features am wahrscheinlichsten ist. Ebenfalls wird definiert, dass diejenigen Parameter, welche für jede Entscheidungslogik variieren können, abrufbar sind.

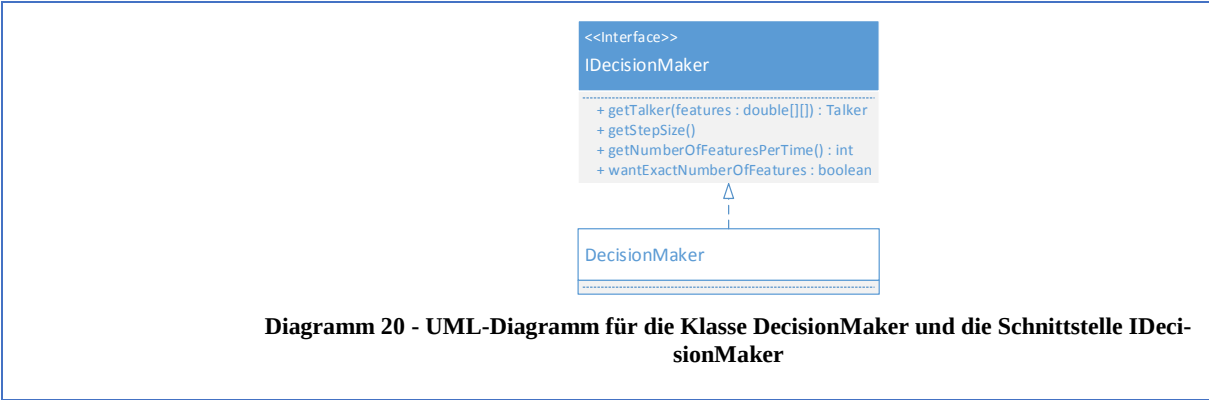


Tabelle 19 - Beschreibung der Klasse DecisionMaker und der Schnittstelle IDecisionMaker

Nachfolgende vier Klassen sind für die Unterhaltung besonders relevant und können nicht ersetzt werden. Sie definieren jedoch keine austauschbaren Komponenten.

Klasse	Beschreibung
Talk	Das „Talk“-Objekt enthält alle Teilnehmer einer Unterhaltung. Es speichert zudem die Daten, welche während der Unterhaltung erfasst werden, etwa wann welcher Sprecher für wie lange gesprochen hat. Dies geschieht mit einer Liste von „Record“-Objekten. Die „Talk“-Klasse implementiert die Schnittstelle „IStateObject“ und definiert somit die Laufzeit von Threads. <pre> classDiagram class Talk { -mTalkers : Talker[] -mRecords : List<Record> +addRecord(talkerId : long, lastTalkerSpeakTime : long) } class Record { -mTalkerId : long -mDuration : long } class IStateObject { <<interface>> +getState() : State +setState() } Talk -- > IStateObject Talk "1" -- "1..*" Record </pre>

Abbildung 15 - UML-Diagramm der Klasse Talk

TalkStatisticData	Speichert statistische Daten über die Sprecher in einer Unterhaltung, so dass nicht alle Daten bei einer Abfrage für die Aktualisierung eines Diagramms neu berechnet werden müssen. <pre> classDiagram class TalkStatisticData { +addChange(newTalker : Talker) +getPercentageTalkTimeOfEachTalker() : HashMap<Long, Double> } class Talk { +addRecord(talkerId : long, lastTalkerSpeakTime : long) } TalkStatisticData --> Talk </pre>
--------------------------	--

Abbildung 16 - UML-Diagramm der Klasse TalkStatisticData

Die Instanz wird in der „Talk“-Activity erzeugt. Von der „TalkAnalysis“-Activity wird sie per Parcel [33] in einem Intent [34] empfangen.

Audio-Ana- lysis- Runnable

Runnable, welches die Zuordnung von Audiodaten zu einem Profil durchführt. Für dies wird eine Klasse benötigt, welche die Schnittstelle „IDecisionMaker“ implementiert. Mit den Rückgabewerten dessen Methoden verarbeitet das Runnable die Daten weiter und speichert sie korrekt in der Klasse „TalkStatisticData“ ab.

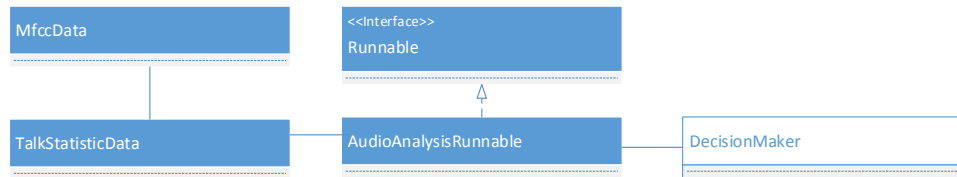


Diagramm 21 - UML-Diagramm für die Klasse AudioAnalysisRunnable

TalkService

Ist ein Service, welcher die rechenintensiven Aufgaben für die Sprechererkennung ausführt. Er verwendet zwei Threads, welche parallel ausgeführt werden. Im einen Thread wird das Runnable „AudioStreamConverterRunnable“ verwendet. Im anderen Thread wird das Runnable „AudioAnalysisRunnable“ ausgeführt. Die Instanz der „TalkService“-Klasse wird von der „Talk“-Activity gestartet.

Tabelle 20 - Beschreibung der Klassen Talk, TalkStatisticData, TalkService und AudioAnalysisRunnable

Die beiden Runnables „AudioStreamConverterRunnable“ und „AudioAnalysisRunnable“ werden von der „TalkThreadManager“-Klasse ausgeführt.

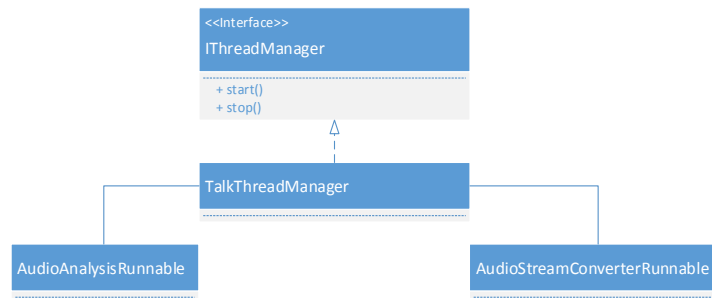


Diagramm 22 - UML-Diagramm der Klasse TalkThreadManager und der Schnittstelle IThreadManager

In nachfolgendem UML-Diagramm sind die Beziehungen der Klassen für eine Unterhaltung zusammengefasst. Das „AudioStreamConverterRunnable“-Objekt stellt das Objekt gemäss Diagramm 14 dar, ergänzt durch das „IStateObject“-Objekt „TalkStatisticData“.

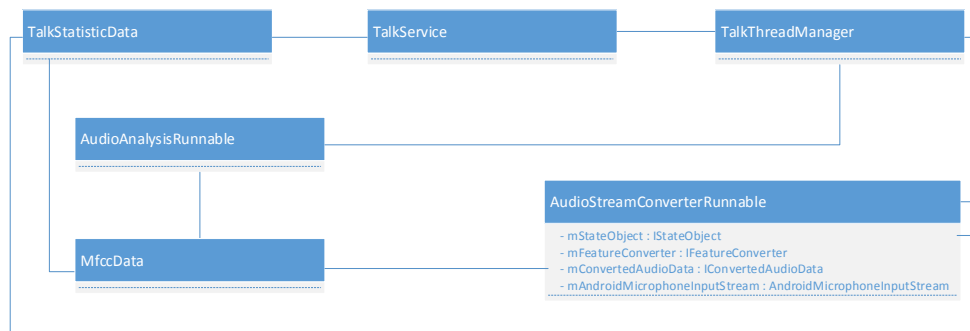


Diagramm 23 - UML-Diagramm Unterhaltung führen

4.2.3 Kalibrierung erweitern

Um eine Kalibrierung zu erweitern werden die gleichen Klassen und Schnittstellen wie für die Kalibrierung verwendet werden. Die Unterschiede sind, dass das „Calibration“-Objekt von der „ExtendCalibration“-Activity erzeugt wird und dass für das Modell nicht nur alle neuen MFCCs verwendet, sondern die bisherigen MFCCs des Modells ebenfalls hinzugenommen werden.

4.2.4 Austauschbare Komponente

Nachfolgend werden jene Klassen und Schnittstellen beschrieben, welche gemäss Kapitel 3.1.7 für eine austauschbare Komponente relevant sind. Es wurden drei Klassen erstellt, in welchen die Parameter für die Audio-Verarbeitung und das Erstellen des Modells zentral verwaltet werden. Da diese Klassen „public static“-Variablen bzw. Methoden besitzen, kann darauf von der ganzen Applikation zugegriffen werden.

Klasse	Beschreibung	Anpassungen
AudioParameters	Enthält diejenigen Konstanten und Methoden, welche für die Audio-Verarbeitung relevant sind.	Hinzufügen oder ändern von Parametern.
ModelParameters	Enthält diejenigen Konstanten, welche für das Sprechermodell relevant sind.	Hinzufügen oder ändern von Parametern.
MfccLibrary-SpecificParameters	Enthält diejenigen Konstanten und Methoden, welche zwischen MFCC-Bibliotheken abweichen können.	Hinzufügen oder ändern von Parametern.

Tabelle 21 - Beschreibung der Klassen **AudioParameters**, **ModelParameters** und **MfccLibrarySpecificParameters**

Nachfolgend werden die weiteren austauschbaren Komponenten beschrieben. Dies soll als Übersicht dienen, für welche Komponente welche Klasse mit welcher Schnittstelle in welcher Klasse ersetzt werden kann. Das genaue Austauschverfahren wird in der Evaluation im Kapitel 5.2 beschrieben.

Komponente	Zu ersetzende Klasse	Relevante Schnittstelle	Klasse mit Schnittstelle
Features extra-hieren	MfccConverterComirva	IFeatureConverter	AudioStreamConverter-Runnable
Filter für Audio-daten hinzufügen oder entfernen	MfccSilenceFilter	IFilterSilence	MfccData
Sprechermodell	GaussianMixtureModel	IDefaultTalkerModel	ModelCreatorRunnable
	Die Komponente für die Entscheidungslogik der Sprecherbestimmung könnte ebenfalls relevant sein, falls keine Log-Likelihood-Werte zurückgegeben werden. In diesem Fall müssten die Werte anders interpretiert werden. Relevant ist zudem die Klasse „TalkerModelStorage“. Diese führt die Serialisierung und Speicherung sowie Deserialisierung und Laden des Sprechermodells durch. Da für die Serialisierung und Deserialisierung die GSON-Bibliothek [35] verwendet wird, muss bei der Deserialisierungs-Methode in der „TalkerModelStorage“-Klasse die Klasse des Sprechermodells angegeben werden. Dies wird benötigt, damit das serialisierte Objekt in der korrekten Klasse geladen wird.		
Entscheidungs-logik für die Sprecherbestimmung	DecisionMaker	IDecisionMaker	AudioAnalysisRunnable

Tabelle 22 - Zugehörigkeiten und Abhängigkeiten der austauschbaren Komponenten

4.3 Sprechererkennung

Es sind alle Komponenten für die Sprechererkennung implementiert. Die Erzeugung der MFCCs wird wie geplant mit der CoMIRVA-Bibliothek durchgeführt. Da jedoch die CoMIRVA-Bibliothek eine Schwäche für das Sprechermodell in dieser Projektarbeit besitzt (siehe Kapitel 3.3.2) wurde auf zwei weitere Technologien ausgewichen (siehe Kapitel 3.3.2). Die Apache Commons Mathematics-Bibliothek [36] verfügt über eine effiziente Berechnung für den k-Means-Algorithmus anhand eines k-Means++-Algorithmus. Die drei Berechnungen von Tabelle 3 sind selbst implementiert, um mit dem Resultat weiterarbeiten zu können. Des Weiteren ist der EM-Algorithmus und die Test-Methode bzw. die Log-Likelihood-Methode für das Sprechermodell anhand vorhandenem C++-Code [16] in Java übersetzt. Nach dem Terminieren des EM-Algorithmus kann es sein, dass unerwünschte Werte errechnet wurden, etwa „Infinty“ oder „NaN“. So ist der Code erweitert worden, dass in diesem Fall die Anzahl gaussische Komponenten schrittweise um eine Komponente solange verringert wird, bis kein solcher Wert errechnet wurde.

Um für ein Sprechermodell möglichst nur die Stimme des Sprechers zu verwenden, anstatt die gesamten Audiodaten, welche auch Stille beinhalten, wird ein Stille-Filter eingesetzt. Dieser wurde selbst implementiert. Ebenfalls wird dieser Filter verwendet, um während einer Unterhaltung die Zuweisung der Audiodaten an das Profil „Niemand“ vorzunehmen. Das Filtern geschieht anhand der MFCCs. Bei der Generierung der MFCCs wird in jedem Fall die erste Dimension der MFCCs für die Lautstärke verwendet. Sind die MFCCs gefiltert, so wird die erste Lautstärke anschliessend für das Modell verworfen, falls dies in den Parametern definiert wurde. Der Stille-Filter verwendet die jeweils 60 kleinsten Lautstärke-Werte und berechnet den Durchschnitt davon. Zum Durchschnitt wird ein bestimmter Wert für die Stille-Grenze addiert. Des Weiteren wird fortlaufend überprüft, ob die neuen MFCCs kleinere Werte für die Lautstärke besitzen. Werden kleinere Werte für Lautstärke eingelesen als die bisherigen, welche den Durchschnitt für die Stille bilden, so wird der Durchschnitt mit den neuen kleineren Werten berechnet.

Nachfolgend eine Übersicht, mit welchen Technologien die Sprechererkennung implementiert wurde.

Komponente	Technologie
Features extrahieren	CoMIRVA-Bibliothek.
K-Means-Algorithmus	Apache Commons Mathematics Library [36] und eigene Implementation.
Sprechermodell mit EM-Algorithmus und Log-Likelihood-Methode	Eigene Implementation anhand von bestehendem C++-Code [16].
Stille-Filter	Eigene Implementation.
Entscheidungslogik für die Sprecherbestimmung.	Eigene Implementation.

Tabelle 23 - Komponenten des Sprechererkennungsalgorithmus und dessen Implementation

Kalibrierung

Die Kalibrierung verläuft gemäss nachfolgendem Diagramm. Der Thread mit dem Runnable „AudioStreamConverterRunnable“ und der Thread mit dem Runnable „ModelCreatorRunnable“ werden nacheinander ausgeführt. Gemäss Tabelle 18 wird das „Calibration“-Objekt dem Kalibrierungsservice mitgegeben und nur für das Runnable „AudioStreamConverterRunnable“ benötigt.

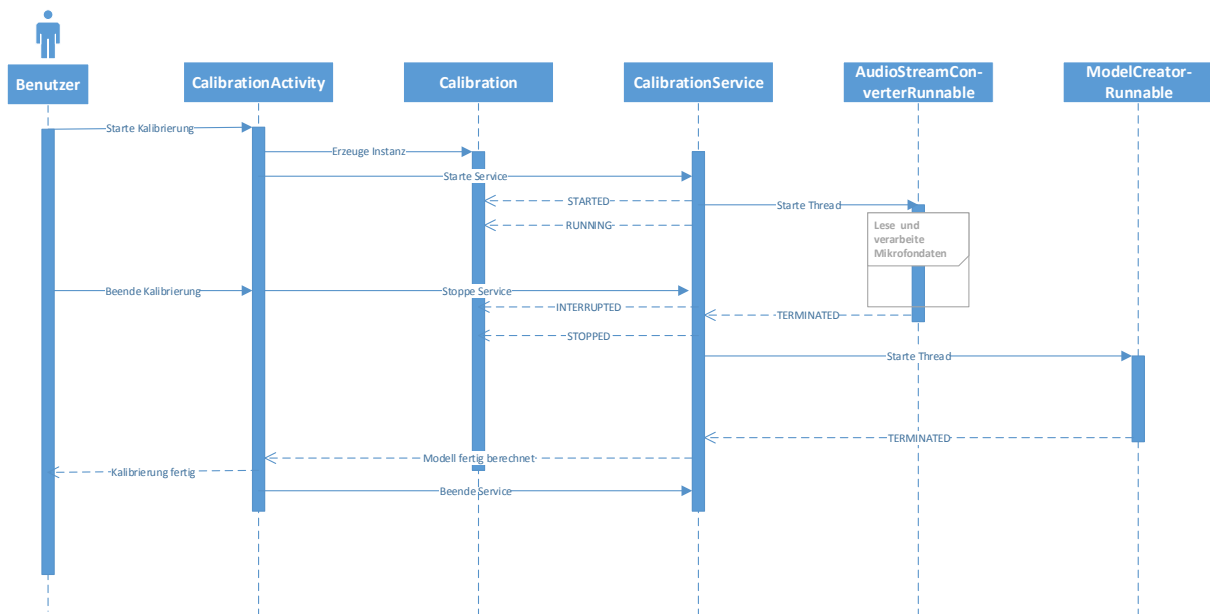


Diagramm 24 - Ablauf Kalibrierung

Ist das Modell erstellt, so wird es dem Benutzerprofil zugewiesen und lokal auf dem Handy gespeichert. Dabei werden die berechneten Variablen des Modells inklusive den trainierten MFCCs gespeichert. Das Speichern der MFCCs ist deshalb notwendig, da für ein erneutes Trainieren alle MFCCs benötigt und verarbeitet werden. Für das Speichern wird das Modell mit GSON [35] zu einem JSON-String verarbeitet und in den Einstellungen der SharedPreferences [37] als Key-Value-Paar im Kontext-Modus „MODE_PRIVATE“ gespeichert. Dieser setzt um, dass die gespeicherten Daten nur für die Applikation zugänglich sind [37]. Somit sind die MFCCs vor unberechtigtem Zugriff geschützt, sofern kein Root-Zugriff erlangt wird.

Je nach Datenmenge und Qualität der Daten variiert die Berechnungszeit für das Sprechermodell. Ebenfalls spielt es eine Rolle, ob Stille gefiltert wird oder nicht. Für eine Minute Sprechzeit ohne nach Stille zu filtern wird das Modell in etwa fünf Sekunden berechnet. Bei drei Minuten ungefilterten Audiodaten, entsteht eine Berechnungszeit von etwa 25 Sekunden. Dies wird nicht als benutzerfreundlich gewertet. Da es jedoch nur beim ersten Applikationsstart durchgeführt werden muss und das Modell gleich nach dem Kalibrieren erstellt werden soll, so dass der Benutzer anschliessend die Applikation verwenden kann, wird diese Zeit vom Benutzer vorausgesetzt.

Unterhaltung führen

Wird eine Unterhaltung gestartet, so erstellt die „Talk“-Activity ein neues „TalkStatisticData“-Objekt. In diesem werden das Sprecherprofil des ausgewählten Benutzers sowie die zwei weiteren Profile „Andere“ und „Niemand“ hinzugefügt. Diese beiden Profile besitzen aktuell kein Sprechermodell. Sie dienen lediglich der Zuordnung, falls der Sprechererkennungsalgorithmus die während der Unterhaltung eingelesenen Audiodaten nicht dem geladenen Sprecherprofil zuordnen kann.

Im Service für die Unterhaltung, dem „Talk“-Service, werden zwei Threads gleichzeitig ausgeführt. Im einen Thread werden fortlaufend die Audiodaten zu MFCCs konvertiert und im „MfccData“-Objekt gespeichert. Im anderen Thread werden kontinuierlich MFCCs aus dem „MfccData“-Objekt ausgelesen und der Entscheidungslogik für die Sprecherbestimmung übergeben. Jeder Sprecherwechsel wird anschliessend im „TalkStatisticData“-Objekt gespeichert. Standardmässig wird alle zehn Sekunden eine Aktualisierung des Diagramms anhand der Daten im „TalkStatisticData“-Objekt mit einem AsyncTask [38] in einem TimerTask [39] von der „Talk“-Activity fortlaufend ausgeführt.

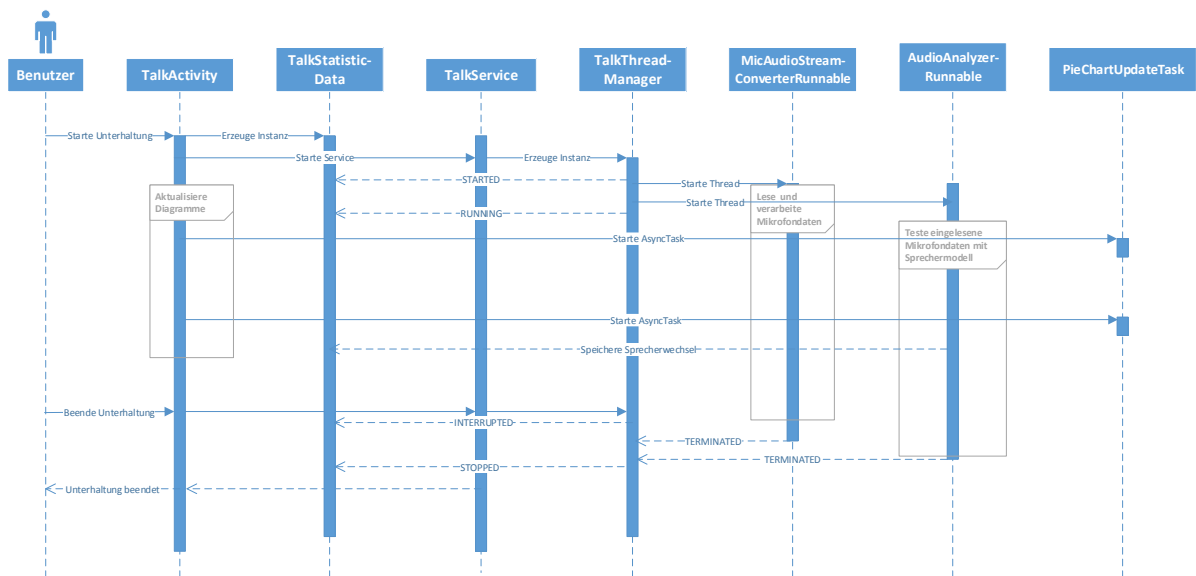


Diagramm 25 - Ablauf Unterhaltung führen

Die Entscheidungslogik für die Sprecherbestimmung entscheidet für die Zuweisung von einer gewissen Anzahl Features zu einem Profil gemäss Diagramm 5. Aus diesen Features wird jeweils pro Dimension ein Durchschnitt gebildet. Diese Durchschnittswerte werden anschliessend in einem neuen MFCC gespeichert. Für das Testen mit dem Modell wird jeweils nur dieses neue MFCC verwendet. Mit diesem Durchschnitt sollen bessere Resultate für die Sprecherbestimmung erreicht werden. Ebenfalls wird in dieser Klasse festgelegt, wie gross die Schrittgrösse für die MFCCs sein soll. Diese bewirkt, welches das nächste MFCC von der Klasse „MfccData“ ist, welches verarbeitet wird. Dieser Wert wurde auf 1 gesetzt, um alle MFCCs zu verwenden.

Die Sprecherbestimmung führt die Entscheidungslogik mit dem Stille-Filter durch. Der Stille-Filter stellt eine Methode zur Verfügung, welche zurückgibt, ob die Audiodaten als Stille gewertet werden oder nicht. Es ist deshalb wichtig, dass bis hierhin die Lautstärke in den MFCCs abgebildet wird. Anschliessend verwirft die Entscheidungslogik diese, falls das Sprechermodell nicht mit der Lautstärke berechnet wurde. Werden die Audiodaten nicht als Stille gewertet, so wird die Wahrscheinlichkeit für die Audiodaten zum Sprechermodell mit der Test-Methode des Sprechermodells berechnet. Ein Wert definiert dabei die Grenze für die Wahrscheinlichkeit, ab wann die berechnete Wahrscheinlichkeit für das Modell genug hoch ist, um die Audiodaten dem Sprechermodell zuordnen zu können. Dies wird aktuell mit einem allgemeingültigen Schwellenwert festgelegt. Jedoch ist dieser Wert abhängig vom erstellten Sprechermodell. Denn der k-Means-Algorithmus startet mit einem zufälligen Startpunkt (siehe Kapitel 2.2.2). Daraus resultieren verschiedene Modelle für die gleichen Daten, wodurch wiederum verschiedene Log-Likelihood-Werte für die Sprechermodelle, erstellt aus den gleichen Daten, erhalten werden. Diese Tatsache wird in der Evaluation (siehe Kapitel 5.3) genauer thematisiert.

Wird die Unterhaltung pausiert, so werden keine Daten mehr aufbereitet. Es werden alle Threads und TimerTasks [39] gestoppt. Das „TalkThreadManager“-Objekt, der „Talk“-Service und das „TalkStatisticData“-Objekt bleiben erhalten, um die Unterhaltung effizient und mit den gleichen Daten fortsetzen zu können.

Kalibrierung erweitern

Ein Benutzer kann sein Sprecherprofil erweitern. Dabei fügt er neue Daten mit dem bereits vorhandenen Profil zusammen. Grundsätzlich ist es derselbe Vorgang wie bei der Kalibrierung, ausser, dass die bereits bestehenden MFCCs geladen und zu den neu erzeugten MFCCs hinzugefügt werden.

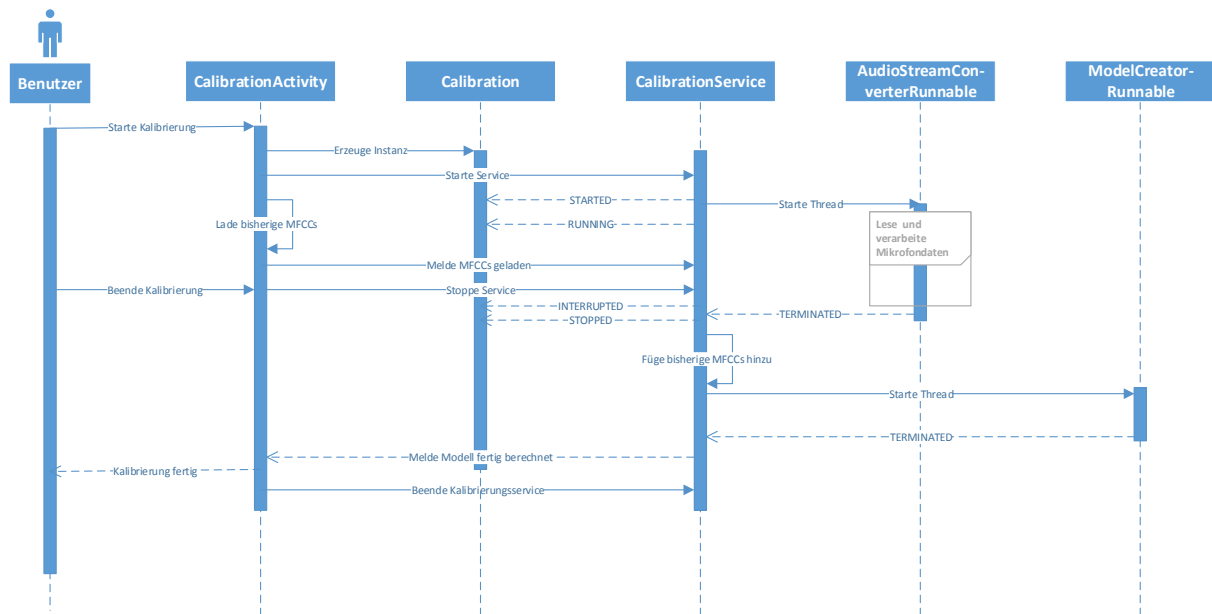


Diagramm 26 - Ablauf Kalibrierung erweitern

5.1 GUI

Das GUI muss unter anderem die Anforderungen definiert in Kapitel 3.1.5 erfüllen. Die Evaluation dieser Kriterien war ein fortlaufender Prozess. Es wurden vier Testpersonen miteinbezogen. Dabei wurde darauf geachtet, dass die Testpersonen kein technisch ausgeprägtes Fachwissen besitzen, um zu gewährleisten, dass der verwendete Wortschatz, Symbole und Abläufe für einen technisch nicht versierten Benutzer verständlich sind. Nachfolgende Personen wurden für die Evaluation miteinbezogen.

Geschlecht	Alter	Beruf
Männlich	26 Jahre	Hochbauzeichner.
Weiblich	25 Jahre	Ausbildung zu Physiotherapeutin.
Männlich	23 Jahre	Ausbildung in Hotelfachschule.
Weiblich	63 Jahre	Pflegefachfrau.

Tabelle 24 - Personen für die Evaluation des GUIs

5.1.1 Vorgehen

Jeder Testperson wurde der Auftrag gegeben, sich in die Situation hineinzusetzen, dass sie eine Applikation auf dem Handy installiert haben mit welcher sie ein Gespräch analysieren können und nun diese ausprobieren wollen. Danach mussten die Testpersonen die Applikation ausführen. Das erste Mal ohne Hilfe, um zu testen, ob der Ablauf verständlich ist. Anschliessend wurde nochmals gemeinsam jeder Schritt bzw. Bildschirmseite besprochen. Dabei wurden Fragen gestellt, welche das Verständnis zeigen soll, wieso sie zum Beispiel die Kalibrierung ausführen müssen. Ebenfalls wurde gefragt, ob sie bei den Bildschirmseiten alles verstanden haben oder wie sie die Diagramme interpretieren. Weiter wurde nachgefragt ob sie Elemente weglassen oder hinzufügen würden.

Während der Evaluierung wurden Notizen gemacht. Diese Notizen wurden anschliessend verwendet, um die wesentlichen Änderungen, welche das Verhalten der Testpersonen beeinflusste, sofort vorzunehmen. Wurden diese Änderungen implementiert, so wurde das GUI der Testperson nochmals vorgelegt, um zu prüfen, ob das Resultat nach den Änderungen der Vorstellung der Testperson entspricht.

Nachdem die Evaluierung mit den vier Testpersonen durchgeführt worden ist, wurden alle Notizen auf Gemeinsamkeiten überprüft. Sind in mehreren Evaluierungen ähnliche Notizen gemacht worden, so wurden diese ebenfalls implementiert.

5.1.2 Ergebnisse

Das GUI musste bei den ersten zwei Testpersonen wesentliche Änderungen erfahren. Einerseits musste der Text angepasst werden, so dass die Testpersonen besser verstehen, was und wieso sie etwas tun. Ebenfalls musste das GUI so angepasst werden, dass grundsätzlich nur ein Benutzerprofil existiert, da sonst Verwirrung entstehen konnte. Des Weiteren musste die Analyse des Gesprächs einfacher gestaltet werden. Dies beinhaltet ebenfalls, dass sie zur Bildschirmseite der Unterhaltung ähnlicher wurde.

Von zwei Testpersonen wird bemängelt, dass zu viel Text vorhanden ist. Es wurde jedoch keine wesentliche Änderung diesbezüglich vorgenommen. Denn es wurde beobachtet, dass während der Kalibrierung mit weniger Einleitungstext zu wenig gesprochen wurde oder nicht genau klar war, wie bei der Kalibrierung vorzugehen ist. Zudem mussten Wörter ersetzt werden, weil sie nicht verstanden wurden.

Ebenfalls wurde auf dieser Bildschirmseite vorgeschlagen, die Bilder automatisch wechseln zu lassen, womit eine Schaltfläche wegfallen würde. Dies wurde ebenfalls nicht umgesetzt, da dem Benutzer die Freiheit gelassen werden soll, selbst zu entscheiden, wann er wechseln will. Dies beruht auch auf der Tatsache, dass beobachtet wurde, dass abhängig von der Testperson verschieden viel zum gleichen Bild gesprochen wird. Würde ein automatischer Wechsel umgesetzt werden sollen, so müsste eine passende Zeit evaluiert werden, nach welcher ein Bilderwechsel vorgenommen wird. Anschliessend könnte mit einem TimerTask [39] der Bilderwechsel durchgeführt werden.

Das Kreisdiagramm wurde in jedem Fall korrekt interpretiert. Das gestapelte Säulendiagramm wurde nur von der Hälfte der Testpersonen richtig interpretiert. Dies jedoch auch nur mit Mühe. Dies ist vor allem darauf zurück zu führen, dass die Ausrichtung der Labels nicht wunschgemäss arbeitet. Eine Möglichkeit, dies zu umgehen wäre gewesen, den Zeitbereich pro Säule im Label anzugeben. Dies würde jedoch zu viel Text verursachen und wäre eher eine Umgehungslösung als eine Verbesserung. Zusätzlich wurde bemängelt, dass die Legende des gestapelten Säulendiagramms schlecht lesbar und die Navigation auf der Zeitachse mit Zoomen in jedem Fall funktionieren müsste.

Da die AChartEngine-Bibliothek durchaus einige entscheidende Schwächen aufweist, könnte gemäss Kapitel 3.4.1 deshalb überprüft werden, ob diese Umstände in der Androidplot-Bibliothek ebenfalls existieren. Falls nicht, wird empfohlen die Diagramme mit der Androidplot-Bibliothek zu realisieren.

5.2 Codestruktur

Um die Codestruktur zu evaluieren, wird das Austauschverfahren der austauschbaren Komponenten aus Kapitel 4.2.4 beschrieben. Das Austauschverfahren für die Komponenten von Tabelle 21 wird aufgrund des trivialen Vorgehens nicht behandelt. Die übrigen Komponenten werden entweder durch eine Komponente von einer anderen Bibliothek oder durch eine neue Klasse, welche eigenen Code besitzt, ersetzt. Das Ziel hierfür ist, dass die Komponenten ausgetauscht werden können, ohne dass die Funktionalität der Applikation beeinträchtigt wird, und der Austauschvorgang einfach auszuführen ist.

Komponente Features extrahieren austauschen

Methode	Die bestehende MFCC-Verarbeitung durch die CoMIRVA-Bibliothek wird durch die ocvolume-Bibliothek [40] ersetzt. Aufgrund der Komplexität der PocketSphinx Android-Bibliothek wird eine unproblematische Bibliothek verwendet.
Vorbedingungen	Ocvolume-Bibliothek importieren.
Auszuführende Schritte	1.1 Neue Klasse „MfccConverterOcv“ erstellen und die Logik für das Verarbeiten von Audiodaten zu MFCCs implementieren. Im Konstruktor wird eine Instanz des „AndroidMicrophoneInputStream“ übergeben (siehe Beschreibung von Diagramm 14). 2. In der Klasse „AudioStreamConverterRunnable“ die Zeile <pre><i>mFeatureConverter = new MfccConverterComirva(mAndroidMicrophoneInputStream);</i></pre> zu <pre><i>mFeatureConverter = new MfccConverterOcv(mAndroidMicrophoneInputStream);</i></pre> ändern.
Bemerkung	Zusätzlich müssen die Parameter und Methoden in der Klasse „MfccLibrarySpecificParameters“ überprüft und falls nötig angepasst werden.

Tabelle 25 - Beschreibung Austauschverfahren für Audio-Merkmale extrahieren

Komponente Sprechermodell austauschen

Methode	Die bestehende Modellerzeugung wird durch die Modellerzeugung mit der CoMIRVA-Bibliothek ausgetauscht.
Vorbedingungen	CoMIRVA-Source-Code so anpassen, dass alle benötigten Klassen auf Android benutzt werden können, etwa javax-Abhängigkeiten entfernen. In der Klasse „ArrayConverter“ wurde die Konvertierung von zweidimensionalen double-Arrays zu einer von der CoMIRVA benötigten „Pointlist“ implementiert.
Auszuführende Schritte	1.1 Neue Klasse „GaussianMixtureModelComirva“ erstellen und die Logik für das Erstellen des Sprechermodells implementieren. 1.2 Über allen zu serialisierenden Variablen, ausser den MFCCs, ist eine Expose-Annotation [35] zu setzen. Dabei ist zu beachten, falls das Modell weitere Objekte enthält, welche serialisiert werden sollen, in diesen ebenfalls die Annotation zu setzen ist. 2. In der Klasse „ModelCreatorRunnable“ die Zeile <code>mDefaultTalkerModel = new GaussianMixtureModel();</code> zu <code>mDefaultTalkerModel = new GaussianMixtureModelComirva();</code> ändern. 3. In der „TalkerModelStorage“-Klasse die Zeile <code>return new Gson().fromJson(PreferenceManager.getDefaultSharedPreferences(context).getString(getTalkerModelSharedPreferencesId(talker), ""), GaussianMixtureModel.class);</code> zu <code>return new Gson().fromJson(PreferenceManager.getDefaultSharedPreferences(context).getString(getTalkerModelSharedPreferencesId(talker), ""), GaussianMixtureModelComirva.class);</code> ändern.
Bemerkung	Standardmässig wird für den k-Means-Algorithmus die Apache Commons Math-Bibliothek eingesetzt. Wird eine andere Implementation eingesetzt, so müssen dafür möglicherweise zusätzliche Klassen oder Methoden implementiert werden. Mit dem k-Means-Algorithmus der CoMIRVA-Bibliothek war dies nicht notwendig.

Tabelle 26 - Beschreibung Austauschverfahren für das Sprechererkennungsmodell für die Kalibrierung

Entscheidungslogik für die Sprecherbestimmung austauschen

Methode	Die bestehende Klasse „DecisionMaker“ wird durch eine neue Klasse ausgetauscht, welche für das Debugging benötigt wurde.
Vorbedingungen	Keine.
Auszuführende Schritte	1. Neue Klasse „DecisionMakerDebugging“ erstellen und Entscheidungslogik implementieren. 2. In der Klasse „AudioAnalysisRunnable“ die Zeile

	<pre>mDecisionMaker = new DecisionMaker(getTalkersWithTalkerModel(), this.mTalkStatisticData.getAudioData().getSilenceFilter());</pre> zu <pre>mDecisionMaker = new DecisionMakerDebugging(getTalkersWithTalker- Model(), this.mTalkStatisticData.getAudioData().getSilenceFilter());</pre> ändern.
Bemerkung	Falls der Stille-Filter benötigt wird, so muss dieser im Konstruktor mitgegeben werden.

Tabelle 27 - Beschreibung Austauschverfahren für die Entscheidungslogik der Sprecherbestimmung

Filter für Audiodaten austauschen

Methode	Es sollen Filter für die MFCCs hinzugefügt oder ausgetauscht werden.
Vorbedingungen	Keine.
Auszuführende Schritte	1. Neue Klasse „MfccSilenceFilterEvaluation“ erstellen und Logik implementieren. 2. In der Klasse „MfccData“ im Konstruktor die bestehende Klasse <pre>mMfccSilenceFilter = new MfccSilenceFilter();</pre> durch <pre>mMfccSilenceFilter = new MfccSilenceFilterEvaluation();</pre> ersetzen. Ebenfalls kann sie dort auch wieder entfernt werden.
Bemerkung	Der Stille-Filter kann in der „MfccData“-Klasse durch die Methode „getSilenceFilter“ geholt werden.

Tabelle 28 - Beschreibung Austauschverfahren für den Filter von Audiodaten

Werden mehrere Komponenten ausgetauscht, wird gemäss den Instruktionen für die einzelnen Komponenten vorgegangen und schrittweise umgesetzt. Es sind keine weitere Klassen zu berücksichtigen.

Ergebnisse

Die einzelnen Komponenten austauschen hat mit dem beschriebenen Vorgehen funktioniert. In jedem Fall haben die neuen Komponenten den Ablauf der Applikation nicht beeinflusst. Die Funktionen der neuen Komponenten konnten störungsfrei verwendet werden.

Die Austauschverfahren werden nicht als grosse Hürde angesehen. Soll das Sprechermodell ausgetauscht werden, so erfordert die GSON-Bibliothek zusätzliche Schritte, nämlich das Setzen der „Expose“-Annotation bei den zu serialisierenden Variablen. Gemäss Android-Entwicklungsseite wird diese Bibliothek jedoch empfohlen [41]. Abgesehen von dieser Eigenschaft, wird die Bibliothek als äusserst praktisch angesehen, da sie einfach verwendet werden kann und alle benötigten Funktionen enthält. Während der Entwicklung entstanden keine Probleme mit der GSON-Bibliothek. Geschwindigkeitsprobleme wurden keine bemerkt, weshalb die GSON-Bibliothek als geeignet angesehen wird.

5.3 Sprechererkennung

Für die Sprechererkennung werden zuerst die implementierten Komponenten evaluiert. Dazu gehören die Konvertierung von Audiodaten zu MFCCs, der k-Means-Algorithmus und die Logik des Sprechermodells. Anschliessend wird evaluiert, wie gut die Sprechererkennung unter verschiedenen Umständen und Parametern trainierte Sprecher erkennt.

5.3.1 Evaluierung der MFCCs, der k-Means- und der Sprechermodell-Implementation

Für die Evaluierung der MFCCs und des Sprechererkennungsalgorithmus wurde das Referenzmodell aus dem Matlab-Code des Talkalyzer-Projekts (siehe Kapitel 1.3) verwendet, von welchem ausgegangen wird, dass es korrekt funktioniert. Zusätzlich wurden eigene Skripts, welche Methoden der Voicebox-Bibliothek aufrufen, geschrieben. Weisen die Evaluierungsergebnisse des Referenzmodells mit dem Code der Talkalyzer-Applikation auf Android hohe Ähnlichkeiten auf, so wird davon ausgegangen, dass die Implementation dieser Arbeit korrekt ist und verwendet werden kann. Verglichen wird mit dem Korrelationskoeffizient für berechnete Log-Likelihood-Werte sowie mit einer grafischen Darstellung von diesen Werten.

Um für das Referenzmodell und die Android-Applikation die gleiche Ausgangslage zu haben, werden auf Android nicht mehr kontinuierlich Audiodaten eingelesen und ausgewertet. Es werden stattdessen WAVE-Dateien verwendet, welche zuvor mit dem Handy generiert wurden. Die beiden Services (siehe Tabelle 18 und Tabelle 20) werden nicht mehr benötigt und deshalb nicht mehr gestartet. Stattdessen werden einzelne Komponenten davon genommen und einzeln aufgerufen. Zudem wurden die Funktionen Schreiben und Lesen von WAVE-Dateien benötigt. Dafür wurden Klassen der simplesound-Bibliothek [42] verwendet. Das GUI wurde ebenfalls erweitert, so dass ausgewählt werden kann, welche Datei wie verarbeitet werden soll. Die nötigen Funktionen und das abgeänderte GUI sind über die „Debugger“-Klasse, welche in diesem Dokument nicht genauer erläutert wird, vorhanden, indem dort der Wert der Variable „ON“ auf „true“ gesetzt wird. Die Parameter wurden in der Talkalyzer-Applikation auf Android wie nachfolgend aufgeführt gesetzt.

Klasse	Parameter
AudioParameters	SAMPLE_RATE = 16000; WINDOW_DURATION = 32; MIN_FREQUENCY = 20.8; MAX_FREQUENCY = 16000.0; MEL_FILTERS = 40; USE_FIRST_COEFFICIENT = false; FILTER_SILENCE = false;
ModelParameters	COMPONENTS = 32;
MfccLibrarySpecificParameters	COEFFICIENTS = 19;
MFCC	powerFFT = new FFT(FFT.FFT_POWER, windowSize, FFT.WND_HANNING);

Tabelle 29 - Parameter in der Applikation auf Android für die Evaluierung

Für die Fourier-Transformation [43], welche für die Generierung der MFCCs gebraucht wird, kann die Domäne für die Filter per Parameter eingestellt werden. Standardmäßig verwendet die CoMIRVA-Bibliothek für das Leistungsspektrum den Parameter „FFT_NORMALIZED_POWER“ mit dem Wert 4. Dieser Wert muss für eine Übereinstimmung mit der Voicebox-Bibliothek auf den Parameter „FFT_POWER“ mit dem Wert 5 geändert werden. In Matlab wurden die Parameter so angepasst, dass sie mit den Parametern der CoMIRVA-Bibliothek übereinstimmen. Die Sample-Rate [44] wird von Matlab automatisch gesetzt. Der Bereich der Filter wird als Bruch verarbeitet und mit der Sample-Rate verrechnet. Die Parameter wurden wie nachfolgend aufgeführt gesetzt.

M-File	Parameter
config	mfcc_frame_size = 32; mfcc_num_cep_coeff = 19; mfcc_coeff_extension = 'Ntpy'; [45] no_of_gaussians = 32;


```
extract_mfcc_features
```

```
filter = 40;  
low_end_of_the_lowest_filter = 0.0013;  
high_end_of_highest_filter = 1;
```

Tabelle 30 - Parameter in der Applikation in Matlab für die Evaluierung

Evaluierung der MFCCs der CoMIRVA-Bibliothek

Um die MFCCs zu evaluieren, wurde der Code so angepasst, dass eingelesene Mikrofondaten in einer WAVE-Datei gespeichert werden. Zudem wurde implementiert, dass WAVE-Dateien gelesen werden können und daraus MFCCs im Matlab-Format abgespeichert werden. Des Weiteren wurde bereits überprüft, dass in Matlab und auf dem Handy gleich viele MFCCs für die gleiche WAVE-Datei erzeugt werden. Zudem wurden die Parameter in beiden Programmen so eingestellt, dass die gleiche Anzahl Komponenten erzeugt wurde. Anschliessend war das Vorgehen gemäss nachfolgender Abbildung bzw. Beschreibung.

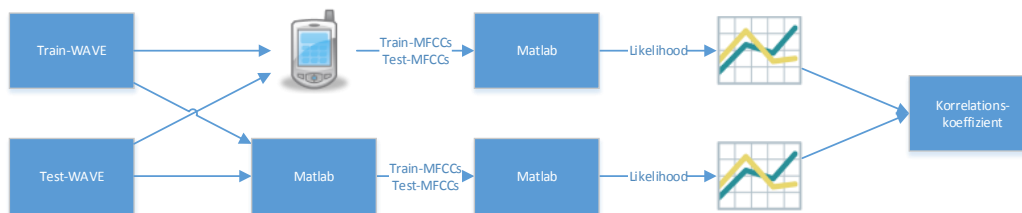


Diagramm 27 – Vorgehen um die MFCCs zu evaluieren

Tätigkeit	Beschreibung
1. Audiodateien erzeugen	Mit dem Handy werden zwei Gespräche anhand von Youtube-Videos aufgenommen. Diese liegen anschliessend als WAVE-Dateien vor. Daraus ergibt sich: 1 Trainings-WAVE-Datei aufgenommen mit dem Handy. 1 Test-WAVE-Datei aufgenommen mit dem Handy.
2. MFCCs mit CoMIRVA generieren	Aus beiden Dateien werden anschliessend MFCCs im Matlab-Format generiert. Dieser Schritt wird ebenfalls auf dem Handy durchgeführt. Daraus ergibt sich: 1 Trainings-MFCC-Datei im Matlab-Format generiert auf dem Handy. 1 Test-MFCC-Datei im Matlab-Format generiert auf dem Handy.
3. Likelihood-Kurve mit CoMIRVA-MFCCs erstellen	Die beiden MFCC-Dateien werden von Matlab eingelesen. Aus der Trainings-MFCC-Datei wird ein Modell in Matlab erstellt. Mit dem Modell und der Test-MFCC-Datei werden die Likelihood-Werte erstellt und die Likelihood-Kurve in Matlab visualisiert.
4. Likelihood-Kurve in Matlab erstellen	In Matlab werden die Trainings-WAVE-Datei und die Test-WAVE-Datei eingelesen und zu MFCCs verarbeitet. Aus den Trainings-MFCCs, generiert in Matlab, wird anschliessend ein Modell erstellt und mit den Test-MFCCs, generiert in Matlab, die Likelihood-Kurve visualisiert.
5. Korrelationskoeffizient berechnen	Um mathematisch zu beweisen, dass die Likelihood-Werte korrelieren, werden die Werte der beiden Likelihood-Werte-Dateien mit der Korrelationskoeffizienten-Funktion „corrcoef“ von Matlab ausgewertet.

Tabelle 31 - Vorgehen um die MFCCs zu evaluieren

Evaluierung der k-Means- und der Sprechermodell-Implementation

Um die k-Means- und die Sprechermodell-Implementation zu evaluieren, musste wieder der Code angepasst werden. Es wurde eine Funktion hinzugefügt, welche es erlaubt, MFCCs im Matlab-Format einzulesen. Zudem müssen die Likelihood-Werte gespeichert und mit einer weiteren Funktion in eine

Datei geschrieben werden, so dass diese in Matlab ausgewertet werden können. Anschliessend war das Vorgehen gemäss nachfolgender Abbildung bzw. Beschreibung.

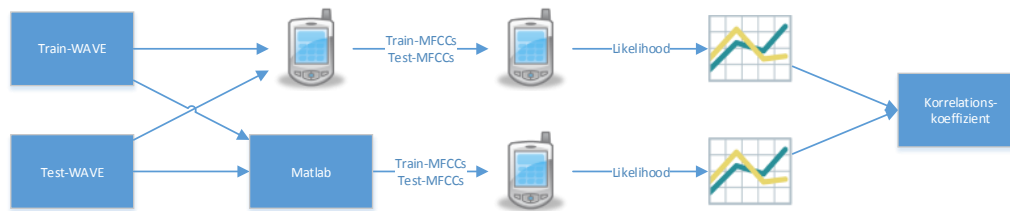


Diagramm 28 – Vorgehen um k-Means- und Sprechermodell-Implementation in der Talkalyzer-Applikation auf Android zu evaluieren

Tätigkeit	Beschreibung
1. Audiodateien bereitstellen	Hierfür werden die bereits erstellten WAVE-Dateien aus Tabelle 31 genommen. Daraus ergibt sich: 1 Trainings-WAVE-Datei aufgenommen mit dem Handy. 1 Test-WAVE-Datei aufgenommen mit dem Handy.
2. MFCCs mit Voicebox generieren	Aus der Trainings-WAVE-Datei wird eine Trainings-MFCC-Datei und aus der Test-WAVE-Datei eine Test-MFCC-Datei in Matlab im Matlab-Format erzeugt. Daraus ergibt sich: 1 Trainings-MFCC-Datei im Matlab-Format generiert in Matlab. 1 Test-MFCC-Datei im Matlab-Format generiert in Matlab.
3. Likelihood-Werte mit Voicebox-MFCCs erstellen	Aus den Trainings-MFCCs wird ein Modell auf dem Handy erzeugt. Mit dem Modell und den Trainings-MFCCs werden die Likelihood-Werte erstellt. Danach werden die Likelihood-Werte in eine Datei geschrieben. Daraus ergibt sich: 1 Likelihood-Werte-Datei generiert aus Matlab-MFCCs.
4. MFCCs mit CoMIRVA generieren	Aus der Trainings-WAVE-Datei und der Test-WAVE-Datei werden die MFCCs im JSON-Format erzeugt, da dies standardmässig in der Talkalyzer-Applikation für die MFCCs verwendet wird. Daraus ergibt sich: 1 Trainings-MFCC-Datei im JSON-Format generiert auf dem Handy. 1 Test-MFCC-Datei im JSON-Format generiert auf dem Handy.
5. Likelihood-Werte mit JSON-MFCCs erstellen	Mit der Trainings-MFCCs wird ein Modell erzeugt. Dabei wurde darauf geachtet, dass das Modell generiert aus den Matlab-MFCCs die gleiche Anzahl Komponenten nach dem Trainieren besitzt, wie das Modell generiert aus den CoMIRVA-MFCCs. Mit dem Modell und den Trainings-MFCCs werden die Likelihood-Werte erstellt. Danach werden die Likelihood-Werte in eine Datei geschrieben. Daraus ergibt sich: 1 Likelihood-Werte-Datei generiert aus den CoMIRVA-MFCCs.
6. Likelihood-Kurven visualisieren	Nun wird die Likelihood-Kurve anhand der Likelihood-Werte-Datei generiert aus Matlab-MFCCs in Matlab visualisiert. Ebenfalls wird die Likelihood-Kurve anhand der Likelihood-Werte-Datei, generiert aus den CoMIRVA-MFCCs, in Matlab visualisiert.
7. Korrelationskoeffizient berechnen	Um mathematisch zu beweisen, dass die Likelihood-Werte korrelieren, werden die Werte der beiden Likelihood-Werte-Dateien mit der Korrelationskoeffizienten-Funktion „corrcoef“ von Matlab ausgewertet.

Tabelle 32 - Vorgehen um k-Means- und Sprechermodell-Implementation in der Talkalyzer-Applikation auf Android zu evaluieren

Ergebnisse

Mit Voicebox und CoMIRVA wird die gleiche Anzahl MFCCs generiert. Zudem werden exakt so viele Likelihood-Werte generiert, wie aus der Test-WAVE-Datei MFCCs generiert werden. Für das Vorgehen aus Tabelle 31 entstehen die nachfolgend visualisierten Log-Likelihood-Kurven.

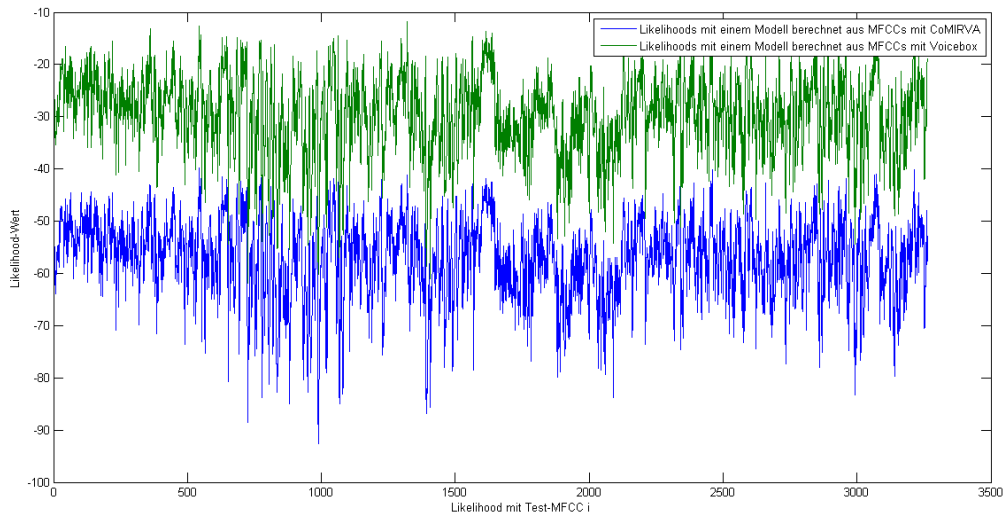


Diagramm 29 – Likelihood-Kurven für das Vorgehen aus Tabelle 27

Der berechnete Korrelationskoeffizient beträgt 0.968. Anhand der Grafik kann herausgelesen werden, dass beide Kurvenverläufe durchaus grosse Ähnlichkeit haben. Als Vergleich wurden zwei Modelle in Matlab generiert, beide aus Voicebox-MFCCs.

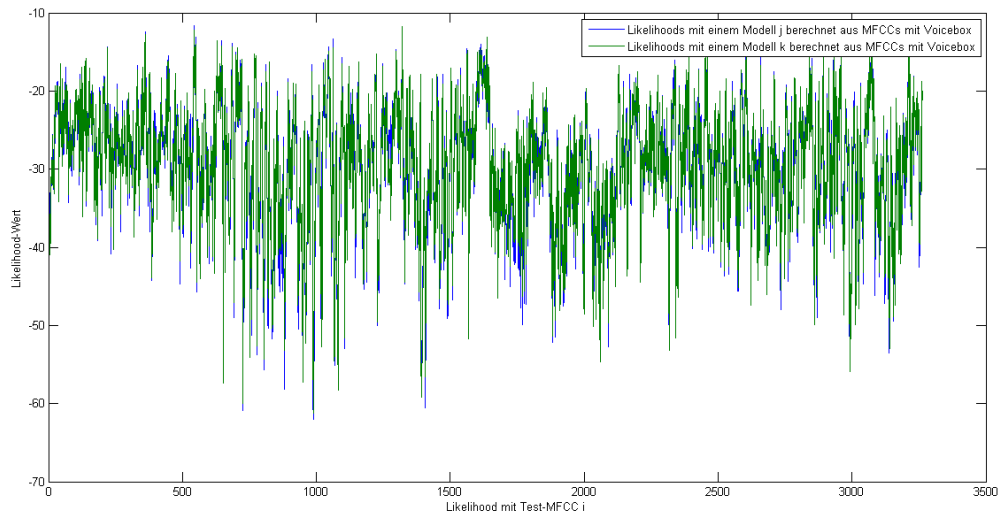


Diagramm 30 – Likelihood-Kurven für zwei verschiedene Modelle generiert aus Voicebox-MFCCs in Matlab

Der berechnete Korrelationskoeffizient beträgt 0.982 und ist somit um den Wert 0.014 höher. Diese Abweichung wird darauf zurückgeführt, dass entweder nicht alle Parameter so gesetzt wurden, dass eine bessere Übereinstimmung zwischen den Voicebox- und den CoMIRVA-MFCCs erreicht wird. Das würde die Parameter aus der Voicebox-Bibliothek und aus der CoMIRVA-Bibliothek betreffen. Andererseits ist es ebenfalls möglich, dass die beiden Bibliotheken mit unterschiedlichen Methoden oder Werten für die gleichen Parameter arbeiten. Da jedoch beide Werte erwartungsgemäss hoch ausfallen und nahe beieinander liegen, wird davon ausgegangen, dass die Implementation der CoMIRVA-Bibliothek korrekt ist. Die Abweichung wurde deshalb nicht weiter untersucht. Für das Vorgehen aus Tabelle 32 entstehen die nachfolgend visualisierten Log-Likelihood-Kurven.

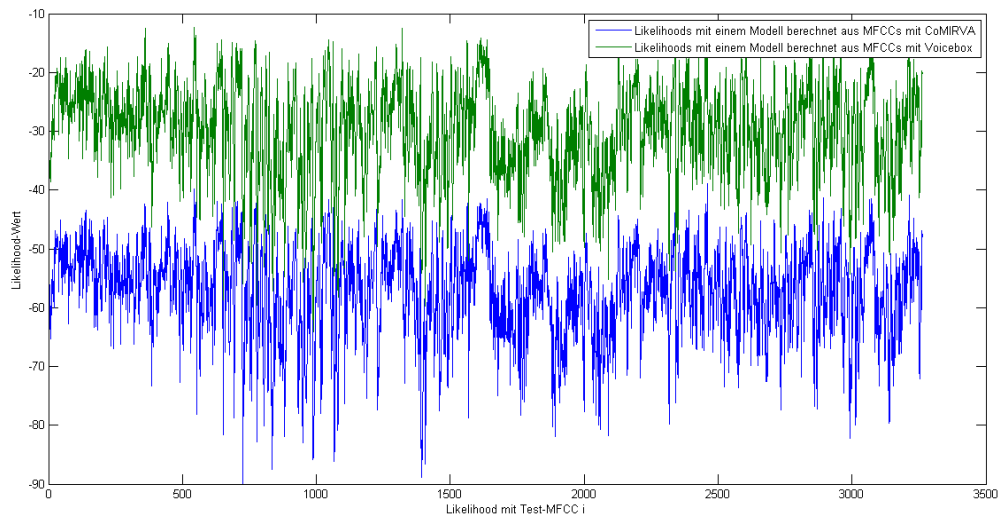


Diagramm 31 - Likelihood-Kurven für das Vorgehen aus Tabelle 28

Der berechnete Korrelationskoeffizient beträgt 0.972. Ebenfalls lässt sich eine hohe Ähnlichkeit beim Verlauf der Kurven erkennen. Ebenfalls wurden zwei Modelle in der Android-Applikation erzeugt, beide aus Trainings-MFCCs generiert mit der CoMIRVA-Bibliothek, bei welchen die Likelihood-Werte anhand der Test-MFCCs, generiert ebenfalls mit der CoMIRVA-Bibliothek, errechnet wurden.

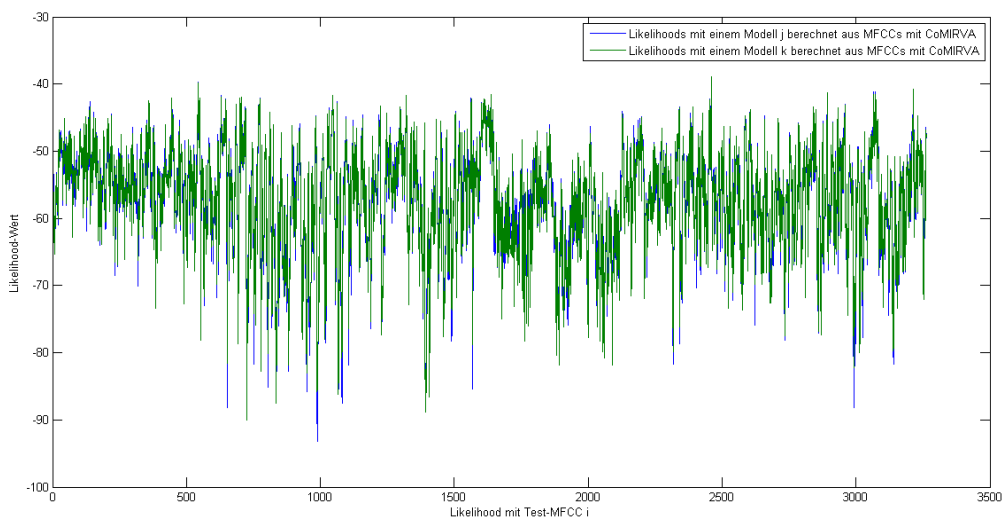


Diagramm 32 - Likelihood-Kurven für zwei verschiedene Modelle generiert aus CoMIRVA-MFCCs auf Android

Der Korrelationskoeffizient beträgt den Wert 0.984. Es lässt sich auch hier eine Differenz feststellen. Beide Korrelationskoeffizienten, generiert mit der Android-Applikation, sind leicht höher.

Anhand der durchaus hohen Ähnlichkeit der Kurvenverläufe mit Modellen erstellt in Matlab und in Android und den ähnlichen Korrelationskoeffizienten wird darauf geschlossen, dass die Implementation mit den definierten Parametern nicht exakt gleich arbeitet, jedoch sehr ähnliche Resultate erzeugt werden. Deshalb wird die Implementation als korrekt und brauchbar gewertet.

5.3.2 Evaluierung der Qualität der Sprechererkennung

Für die Evaluierung der Qualität der Sprechererkennung wird ein Szenario verwendet, welches angenommene typische Eigenschaften für den Einsatz dieser Applikation besitzt. Dazu gehört, dass das Handy in einer Distanz von etwa eineinhalb Metern platziert wird. Dies wird nicht als Distanz zu allen Gesprächsteilnehmern angenommen, sondern wird als die weiteste Distanz eines Teilnehmers betrachtet. Es wird davon ausgegangen, dass ein Benutzer nicht speziell darauf achtet, wie er das Mikrofon des Handys ausrichtet. Deshalb wird für die Evaluierung vorerst das Mikrofon des Handys entgegen der

Richtung des Sprechers ausgerichtet, da angenommen wird, dass unter diesem Umstand schlechtere Resultate erfolgen können als wenn das Mikrofon in die Richtung des Sprechers gerichtet ist. Nachfolgend soll ausgewertet werden, wie gut sich die Sprechererkennung unter diesen Umständen verhält.

Die Parameter wurden gemäss einer ähnlichen Arbeit [24] gesetzt, weil deren Evaluation umfangreiche Parametertests beinhaltet. Als Parameter wurden unter anderem 16 Komponenten für das GMM verwendet und die Anzahl MFCCs zum Testen mit dem Modell wurde auf 190 MFCCs definiert. Dies entspricht ungefähr der Zeit von drei Sekunden. Ebenfalls werden Gruppen von je drei Personen erstellt.

Um die Resultate auswerten zu können, werden Visualisierungen in Matlab und das F-Mass (englisch F1 score) gewählt. Die Visualisierungen in Matlab dienen dazu, die erhaltenen Log-Likelihood-Werte während des Zeitverlaufs auswerten zu können bzw. um die Werte erhalten durch das F-Mass zu bewerten. Das F-Mass bewertet die Genauigkeit und die Trefferquote und dient zur Auswertung dieser anhand einer Kennzahl. Die Werte sind dabei im Bereich von 0 bis 1, wobei 0 als schlechteste Wertung und 1 als beste Wertung angesehen wird [46].

Legende für F-Mass		
	Bedingung positiv	Bedingung negativ
Testergebnis positiv	True positive (TP)	False negative (FN)
Testergebnis negativ	False negative (FN)	True negative (TN)

Tabelle 33 - Legende für das F-Mass

Die Berechnung des F-Mass wurde in Java implementiert und berechnet sich gemäss nachfolgender Formel [46].

$$F = \frac{2 * TP}{2 * TP + FP + FN}$$

Kalibrierungsdatei und simulierte Unterhaltung

Es wurde eine Audiodatei verwendet. Darin enthalten sind die Stimmen von fünf Personen. Das Vorgehen für die Dateien war folgendermassen. Es wurde für jede Person eine Kalibrierung von etwa sechs Minuten durchgeführt. Zusätzlich wurden die Personen für nochmals drei Minuten aus einer Distanz von eineinhalb Metern aufgenommen, das Mikrofon war entgegen der Richtung des Sprechers gerichtet. Anschliessend wurde aus den Kalibrierungsdaten die ersten fünf Minuten für die Trainingsdaten des jeweiligen Sprechers verwendet. Weitere 25 Sekunden wurden verwendet, um die Datei zu bilden, welche als Richtwert für die Log-Likelihood-Werte dienen soll (siehe Abschnitt „Ansatz Grenzwert mit Perzentil bestimmen“).

Die Kalibrierungsdaten sind jeweils unter verschiedenen Umständen aufgenommen worden. Die simulierte Unterhaltung wurde im gleichen Raum unter den gleichen Bedingungen von Distanz und Lage des Mikrofons aufgenommen. Bei den Kalibrierungsdaten sowie bei der simulierten Unterhaltung wurde den Testpersonen nicht vorgegeben, wie sie sich verhalten sollen. Ebenfalls wurden nachträglich auch keine Nebengeräusche entfernt. In den erfassten Audiodaten sind unter anderem vorbeirennende kreischende Kinder, Niesen, Lachen, mit dem Schlüssel spielen, nervöses Sitzen oder die Nase hochziehen vorhanden.

Anschliessend wurde für die Dateien die Stille, welche länger als 100 Millisekunden dauert, mit dem Programm „Audacity“ [47] entfernt. Die Evaluation des Stille-Filters wird im Kapitel 6.2.4 thematisiert.

Audiodatei 1

Ziel Anhand dieser Datei soll erkannt werden, wie gut die Sprecher mit den in diesem Kapitel beschriebenen Bedingungen unterschieden werden können.

Sprecherwechsel Die Sprecher werden nacheinander platziert.

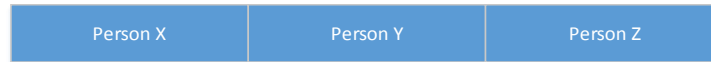


Abbildung 17 - Sprecherwechsel bei der Audiodatei 1

Tabelle 34 - Beschreibung des Aufbaus der Audiodatei 1

Ansatz Grenzwert mit Perzentil bestimmen

Ein bestehendes Problem ist, dass die Log-Likelihood-Werte für jedes Modell variieren (siehe Kapitel 0 im Abschnitt „Unterhaltung führen“). Ein allgemeingültiger Schwellenwert wird als problematisch angesehen, weil der Schwellenwert nicht auf die jeweiligen Modelle abgestimmt ist und deshalb schlechtere Resultate erzielt werden können. Es wird deshalb nachfolgend ein Ansatz überprüft, welcher dieses Problem beheben könnte.

Bisher wurden alle eingelesenen Audiodaten der Kalibrierung für das Sprechermodell verwendet. In der Evaluation wird der Ansatz verfolgt, einen grossen Teil der Trainingsdaten weiterhin für das Sprechermodell zu verwenden. Ein kleiner Teil soll jedoch nicht dafür verwendet werden. Stattdessen werden die Features davon, nachdem das Sprechermodell erstellt wurde, mit dem erstellten Sprechermodell getestet, um die Log-Likelihood-Werte dafür zu erhalten. Daraus resultiert, dass die erwarteten Log-Likelihood-Werte für die Stimme der Person mit dem Sprechermodell erhalten werden. Es lässt sich dadurch ermitteln, in welchem Bereich die Werte ungefähr liegen, falls Audiodaten des Sprechers mit seinem Modell getestet werden. Diese Schritte werden manuell durchgeführt. Ein automatisierter Ablauf wird in Kapitel 6.2.4 thematisiert.

Mit den erhaltenen Log-Likelihood-Werten wird das F-Mass für verschiedene Perzentile berechnet. Ein Perzentil bestimmt dabei ein Hundertstelwert von der gesamten Menge. Die Menge macht in diesem Fall die erhaltenen Log-Likelihood-Werte, erhalten durch das Testen vom kleinen Ausschnitt der Trainingsdatei, aus. Die zu testenden Perzentile sind im Bereich 5 bis 100 in Fünfer-Abschnitte unterteilt. Dies hat den Grund, da zuerst ermittelt wird, wie hoch pro Perzentil die Werte für das F-Mass resultieren. Deshalb wird bei den Ergebnissen das Perzentil ebenfalls angegeben. Es ist jedoch anzumerken, dass nicht der bestmögliche Wert für das F1-Mass berechnet wird, sondern nur der bestmögliche Wert für die definierten Perzentile. Um dies zu korrigieren, müssten kleinere Abstufungen als Fünfer-Werte gemacht werden.

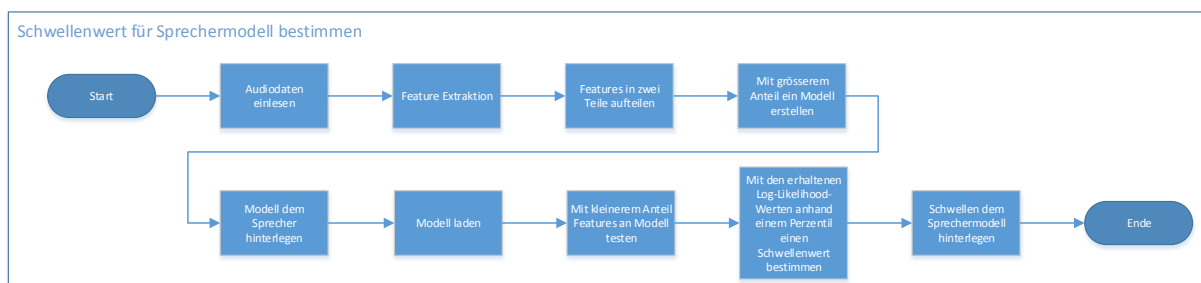


Diagramm 33 – Schwellenwert für ein Sprechermodell bestimmen

Ergebnisse für die Trainingszeit

In den nachfolgenden Tabellen sind die Resultate für das höchste F-Mass mit einer bestimmten Trainingszeit enthalten. Dabei werden pro Person das Perzentil an erster Stelle, der daraus resultierende

Schwellenwert an zweiter Stelle und der Wert für das F-Mass an dritter Stelle angegeben. Es wird die Auswirkung der Trainingszeit für Gruppen von drei Personen aufgezeigt.

Gruppe 1			
Trainingszeit	Person 1	Person 2	Person 3
0.5min	45 / -36.22 / 0.56	100 / -40.62 / 0.49	95 / -40.14 / 0.53
1min	55 / -36.07 / 0.55	85 / -41.77 / 0.51	95 / -40.52 / 0.54
3min	95 / -38.84 / 0.51	80 / -40.53 / 0.50	85 / -38.94 / 0.53
5min	70 / -36.57 / 0.54	90 / -41.56 / 0.50	100 / -40.81 / 0.51

Tabelle 35 - Resultate für verschiedene Trainingszeiten bei Gruppe 1

Gruppe 2			
Trainingszeit	Person 1	Person 4	Person 5
0.5min	35 / -36.08 / 0.73	100 / -38.35 / 0.62	100 / -42.39 / 0.44
1min	65 / -36.55 / 0.80	100 / -37.56 / 0.58	100 / -42.98 / 0.47
3min	80 / -37.21 / 0.75	100 / -37.74 / 0.61	100 / -41.86 / 0.43
5min	45 / -35.56 / 0.70	100 / -36.97 / 0.46	100 / -41.48 / 0.42

Tabelle 36 – Resultate für verschiedene Trainingszeiten bei Gruppe 2

Auffällig ist, dass bei Person vier in der Gruppe zwei eine wesentliche Verschlechterung von drei zu fünf Minuten Trainingsdaten entsteht. In den Trainingsdaten dieser Person konnte jedoch nichts Auffälliges erkannt werden. Solche Ausreisser müssen deshalb in weiteren Tests genauer untersucht werden. Da die Person fünf deutlich schlechtere Resultate aufweist, wurde diese Person in der Audiodatei an eine andere Stelle verschoben, um einen möglichen Implementationsfehler aufzudecken. Damit resultieren jedoch dieselben Resultate. Ebenfalls wurden andere Ausschnitte der Kalibrierungsdaten verwendet, jedoch ohne Verbesserung. Somit wurde die Person fünf nochmals für einen Vergleich mit den beiden anderen Personen gebraucht.

Gruppe 3			
Trainingszeit	Person 2	Person 3	Person 5
0.5min	100 / -40.58 / 0.65	95 / -39.76 / 0.73	100 / -42.39 / 0.45
1min	100 / -40.89 / 0.64	40 / -37.52 / 0.77	100 / -41.07 / 0.41
3min	80 / -40.53 / 0.67	100 / -40.32 / 0.71	80 / -39.62 / 0.60
5min	95 / -41.11 / 0.66	95 / -40.10 / 0.74	85 / -39.94 / 0.60

Tabelle 37 - Resultate für verschiedene Trainingszeiten bei Gruppe 3

Nun weisen die Werte für die Person fünf ab einer Trainingszeit von drei Minuten ähnlichere Werte zu den anderen Sprechern auf. In diesem Fall scheinen die Trainingsdaten dieser Person für eine Unterscheidung der Stimmen in dieser Gruppe erst ab drei Minuten zu genügen. Die Werte der Person zwei und drei sind etwas höher als in der Gruppe 1, womit erkannt werden kann, dass in dieser Gruppe ansatzweise mit den erstellten Modellen zwischen den Personen unterschieden wird. Die Durchschnitte für die erhaltenen Werte des F-Masses sind nachfolgend aufgeführt. Anhand des Durchschnittes ist die beste Trainingszeit drei Minuten.

0.5min	1min	3min	5min
0.578	0.586	0.590	0.570

Tabelle 38 - Durchschnittswerte für das F-Mass mit verschiedenen Trainingszeiten

Nicht erwartet wurde, dass für mehr Trainingszeit, teils schlechtere bzw. nicht immer bessere Resultate und im Durchschnitt sogar das schlechteste Resultat erzielt wurde. Es wurde in keinem Fall einen höchsten Wert mit fünf Minuten Trainingsdaten erreicht, nur in einem Fall gleich hoch wie mit drei Minuten. Dies könnte unter anderem damit zusammenhängen, dass das Beschreiben der Bilder gemäss Testpersonen ab etwa zwei Minuten als anstrengend empfunden wird. Entsprechend haben sich die Testpersonen und deren Stimme teilweise verhalten. Es lässt sich ebenfalls erkennen, dass bereits ab einer Trainingszeit von 30 Sekunden ähnliche Werte erreicht werden. Eventuell würden weitere Tests dies bestätigen und die Kalibrierungszeit könnte verkürzt werden.

Die erhaltenen Werte werden als zu tief bewertet. Für die angedachte Trainingszeit von einer Minute (siehe Kapitel 3.1.1) sind Werte zwischen 0.41 und 0.80 erreicht worden. Es kann also nicht konstant erkannt werden, welcher Sprecher dass spricht. Grundsätzlich scheint im Durchschnitt eine Trainingszeit von einer Minute zu genügen. Nur in einem Fall führte eine höhere Trainingszeit zu einer wesentlichen Verbesserung.

Sprecherwechsel

Um eine Unterhaltung zu simulieren, wurde eine weitere Audiodatei erstellt. Diese enthält die gleichen Audiodaten wie bei der Audiodatei 1. Jedoch wird alle zehn Sekunden ein Sprecherwechsel erzeugt. Anschliessend werden die Veränderungen der Resultate im Vergleich zu den im oberen Abschnitt erhaltenen Resultaten beurteilt.

Audiodatei 2	
Ziel	Anhand dieser Datei soll erkannt werden, wie sich die Resultate mit den gesetzten Parametern bei mehreren Sprecherwechseln im Vergleich zu nur einem Sprecherwechsel verhält.
Sprecherwechsel	Alle zehn Sekunden wird ein Sprecherwechsel erzeugt. Die Personen wiederholen sich.  Abbildung 18 - Sprecherwechsel bei der Audiodatei 2

Tabelle 39 - Beschreibung des Aufbaus der Audiodatei 2

Nachfolgend werden die Resultate mit den gleichen Gruppen wie bisher, jedoch mit der Audiodatei 2 berechnet, dargestellt. Dabei steht bei jeder Person links derjenige Wert, welcher neu erhalten wurde und rechts derjenige aus dem Resultat für die Audiodatei 1.

Gruppe			
1	Person 1	Person 2	Person 3
	0.51 / 0.51	0.50 / 0.50	0.50 / 0.53
2	Person 1	Person 4	Person 5
	0.67 / 0.75	0.59 / 0.61	0.40 / 0.43
3	Person 2	Person 3	Person 5
	0.62 / 0.67	0.67 / 0.71	0.45 / 0.60

Tabelle 40 - Resultate für die Audiodatei 2

Die Resultate führen zu teilweise wesentlichen Verschlechterungen. Nachfolgend sind die Log-Likelihood-Werte der Personen eins und vier von der Gruppe zwei in Matlab visualisiert. Die markierten Bereiche stellen die jeweiligen Zeitabschnitte dar, in denen die Person gesprochen hat.

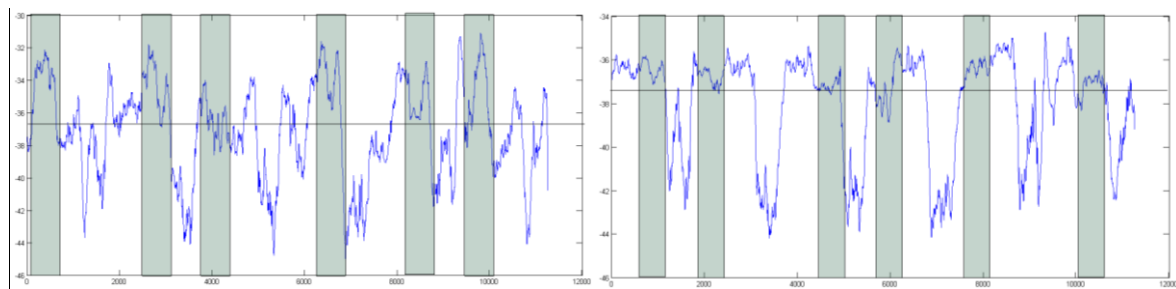


Diagramm 34 – Log-Likelihood-Werte von Person 1 (links) und Person 4 (rechts)

Zu sehen ist, dass bei Person eins die Person vier deutlich besser unterschieden werden kann, als bei Person vier die Person eins. Bei den berechneten Log-Likelihood-Werten des Sprechermodells der Person vier sind die Werte für die Person eins etwa im gleichen Bereich, wie für die Person vier, teils sogar höher. Deshalb wird vermutet, dass in den Trainingsdaten der Person eins deutlich bessere Stimmmerkmale erfasst werden konnten. Person fünf hingegen wird von beiden Sprechermodellen deutlich unterschieden.

Wie zu erkennen ist, entsteht für einen Sprecherwechsel meist eine Steigung oder eine Senkung. Je höher diese Steigung oder Senkung ist, desto deutlicher wird ein Sprecherwechsel wahrgenommen. Wenn viele Sprecherwechsel vorkommen, so resultiert dies jedoch in weniger eindeutig erkennbare Abschnitte für das Sprechermodell, weil aus Features von zwei Sprechern ein Durchschnitt berechnet wird. Somit steigt die Fehleranfälligkeit in diesen Abschnitten zusätzlich. Weiter müsste überprüft werden, ob der Parameter, welcher die Anzahl MFCCs für den Durchschnitt bestimmt, mit anderen Werten zu verbesserten Resultaten führt. Ebenfalls in welchen Zeitabschnitten ein Sprecherwechsel in einer Unterhaltung zu erwarten ist.

Ergebnisse für den Schwellenwert mit einem Perzentil bestimmen

Nun soll evaluiert werden, ob ein Perzentil festgelegt werden kann, mit welchem für mehrere Modelle gute Resultate bezüglich F-Mass erzielt werden können. Nachfolgende Werte resultierten mit denjenigen Sprechermodellen, welche mit drei Minuten Trainingsdaten erstellt wurden. Bei jeder Person stehen links der aus dem Perzentil resultierte Schwellenwert und rechts der Wert für das F-Mass. Ebenfalls wird als Richtwert nochmals angegeben, was das beste Resultat mit Fünfer-Schritten im Bereich von fünf bis hundert gewesen ist.

Perzentil	Person 1	Person 2	Person 3	Person 4	Person 5
60	-36.09 / 0.36	-39.80 / 0.50	-37.62 / 0.24	-36.13 / 0.20	-38.63 / 0.07
70	-36.44 / 0.38	-40.11 / 0.51	-38.06 / 0.44	-36.30 / 0.28	-39.22 / 0.13
80	-37.21 / 0.52	-40.53 / 0.51	-38.78 / 0.51	-36.59 / 0.41	-39.62 / 0.18
85	-37.43 / 0.56	-40.77 / 0.50	-38.94 / 0.50	-36.78 / 0.48	-39.95 / 0.23
90	-37.90 / 0.58	-41.03 / 0.50	-39.21 / 0.49	-37.03 / 0.53	-40.80 / 0.34
94	-38.50 / 0.54	-41.20 / 0.50	-39.84 / 0.50	-37.30 / 0.57	-40.96 / 0.36
98	-38.89 / 0.54	-41.35 / 0.50	-41.19 / 0.51	-37.46 / 0.60	-41.17 / 0.38
100	-39.01 / 0.53	-41.49 / 0.50	-41.34 / 0.51	-37.74 / 0.61	-41.86 / 0.43
Richtwert	0.58	0.51	0.51	0.61	0.43

Tabelle 41 - Werte für das F-Mass für diverse Perzentile

Zu erkennen ist, dass auch vom Perzentil 98 zum Perzentil 100 der Schwellenwert noch deutliche Abweichungen erfahren kann. Grundsätzlich wird mit dem Perzentil 100 einen Wert nahe am Richtwert erreicht. Anhand von diesen Resultaten wird deshalb das Perzentil 100, um einen Schwellenwert für jedes Modell zu berechnen, vorgeschlagen. Es ist jedoch zu beachten, dass für gute Sprechermodelle, welche gewisse Personen unterscheiden können, zum Beispiel jenes von Person eins, ein tieferes Perzentil als 100 zu einem besseren Resultat führt.

Weitere Untersuchungen

Weiter wurde untersucht, wie sich die Werte für das F-Mass verhalten, wenn mehr als drei Personen an einer Unterhaltung teilnehmen. Ebenfalls wird untersucht, wie sich die Lage des Mikrofons auswirkt. Für dies wurden die Personen erneut aufgenommen, das Mikrofon ist nun aus einer Distanz von eineinhalb Metern in die Richtung des Sprechers gerichtet. Es werden keine künstlich erzeugten Sprecherwechsel eingefügt, weshalb der Aufbau gleich zur Audiodatei 1 ist.

Person	Mikrofon entgegen der Richtung des Sprechers gerichtet	Mikrofon in die Richtung des Sprechers gerichtet	Wert von F-Mass für Person in Gruppe von drei Personen
1	95 / -38.68 / 0.51	70 / -36.36 / 0.55	0.51 und 0.75
2	80 / -40.53 / 0.50	40 / -39.55 / 0.47	0.50 und 0.67
3	80 / -38.78 / 0.49	100 / -39.99 / 0.48	0.53 und 0.71
4	100 / -37.44 / 0.39	100 / -37.44 / 0.28	0.61
5	100 / -41.61 / 0.28	100 / -41.61 / 0.34	0.43 und 0.60

Tabelle 42 - F-Mass bei fünf Personen pro Gruppe und gegen den Sprecher gerichtetes Mikrofon

Die Werte sind teils deutlich schlechter als bei den Gruppen von drei Personen. Dies ist darauf zurückzuführen, dass mehr „False negative“-Werte (siehe Tabelle 33) möglich und entstanden sind. Jedoch weichen die Werte bei vier von fünf Personen bei den beiden Ausrichtungen des Mikrofons nicht wesentlich voneinander ab, weshalb davon ausgegangen wird, dass ein entgegen der Richtung des Sprechers gerichtetes Mikrofon aus dieser Distanz nicht entscheidend beeinflusst.

6.1 Zusammenfassung

Die Aufgabenstellung bestand darin, eine Applikation auf Android zu entwickeln, welche einen Sprecher in einer Unterhaltung erkennt. Ebenfalls soll erkannt werden, ob jemand anders oder niemand spricht. Das GUI muss für den Benutzer verständlich sein sowie fortlaufend die Sprecheranteile mit einem Diagramm visualisieren. Um die Sprechererkennung verbessern zu können, soll für einen Software-Entwickler eine Architektur aufgebaut werden, welche erlaubt, einfach Komponenten austauschen zu können.

Das GUI wurde mit den Komponenten realisiert, welche Android zur Verfügung stellt. Um die Diagramme darzustellen wurde aufgrund von einer grösseren Verbreitung und auf die Aufgabenstellung ausgerichteten Beispielen die AChartEngine-Bibliothek verwendet. Um das GUI benutzerfreundlich zu entwickeln, wurden fortlaufend Evaluierungen mit Testpersonen durchgeführt. Die Navigation und der Ablauf sind für die Testpersonen verständlich. Das gestapelte Säulendiagramm ist jedoch entweder schlecht oder überhaupt nicht interpretierbar. Diese Problematik konnte mit der AChartEngine-Bibliothek nicht behoben werden. Es wird deshalb vorgeschlagen, die Androidplot-Bibliothek für eine Implementation in Erwägung zu ziehen.

Für die Softwarearchitektur wurden austauschbare Komponenten definiert, welche zum Ziel haben, einfach ausgetauscht werden zu können. Für diese Komponenten wurde eine Anleitung geschrieben, wie sie auszutauschen sind. In der Evaluation konnten alle Komponenten ausgetauscht werden, ohne dass die Funktionalität der Applikation beeinflusst wurde.

Geplant war, die Sprechererkennung mit der CoMIRVA-Bibliothek zu implementieren. Die CoMIRVA-Bibliothek beinhaltet die Konvertierung der Audiodaten zu MFCCs, den k-Means-Algorithmus sowie ein EM-Algorithmus und die Log-Likelihood-Funktion. Aufgrund von Geschwindigkeitsproblemen mit dieser Bibliothek betreffend Sprechermodell erzeugen, welche nicht behoben werden konnten, wurden andere Technologien eingesetzt. Für die Aufbereitung der Audiodaten zu MFCCs wird weiterhin die CoMIRVA-Bibliothek verwendet. Für den k-Means-Algorithmus ist die Apache Commons Math-Bibliothek sowie eigener Code implementiert. Für den EM-Algorithmus und die Log-Likelihood-Funktion wurde bestehender C++-Code in Java übersetzt. Ein Sprechermodell kann je nach Datenmenge und Qualität in wenigen Sekunden berechnet werden. Die Log-Likelihood-Berechnung von MFCCs zu einem Sprechermodell kann kontinuierlich und mit den jeweils aktuellsten Daten durchgeführt werden. Für eine Trainingszeit von einer Minute werden Werte für das F-Mass zwischen 0.41 und 0.80 erhalten. Die erhaltenen Resultate sind somit nicht befriedigend und werden als verbesserungswürdig angesehen.

6.2 Taktischer Ausblick

Für das GUI und den Sprechererkennungsalgorithmus werden mehr Testdaten bzw. Testpersonen benötigt. Mit diesen können anschliessend umfassende Tests durchgeführt werden um Schwachstellen und Stärken aufzudecken und entsprechend zu behandeln.

6.2.1 Fehler in der Software

Nach einer Kalibrierung wird das berechnete Sprechermodell dem Benutzer hinterlegt. Ebenfalls werden die MFCCs abgespeichert. Diese werden mit einem AsyncTask [38] in die SharedPreferences [37] geschrieben. Wird während dieser Zeit das GUI bedient, kann je nach Datenmenge und Gerät ein Applikationsabsturz vorkommen. Dasselbe gilt beim Löschen eines Profils. Dies konnte aus zeitlichen Gründen nicht weiter untersucht werden. Mit weiteren Tests könnten eventuell weitere Fehler in der Applikation erkannt und anschliessend behoben werden.

6.2.2 GUI

Das GUI wird schlicht gehalten. Ebenfalls wurden grösstenteils Standardkomponenten von Android eingesetzt. Dies könnte durch einen Designer weiterentwickelt werden, um dem Benutzer ein attraktiveres GUI zu bieten. Des Weiteren wurde kein Icon verwendet, durch welches sich die Applikation identifizieren lässt. Ein Auftrag wurde früh genug aufgegeben, jedoch kein Resultat erhalten. Dieses Icon könnte auch im GUI eingesetzt werden, etwa als Start-Schaltfläche, um eine Unterhaltung zu starten.

Um das GUI gemäss den Kriterien aus Kapitel 3.1.5 repräsentativ als erfolgreich werten zu können, müssen weitere Testpersonen miteinbezogen werden. Möglicherweise würde sich daraus ergeben, dass beim Kalibrieren ein automatischer Bilderwechsel von mehreren Personen als besser angesehen wird. Um dies zu realisieren, müsste eine Zeit evaluiert werden, wie lange ein Bild dem Benutzer angezeigt wird.

Verschiedene Bildschirmgrössen

Die Applikation wurde auf dem Gerät gemäss Kapitel 3.1 entwickelt. Aktuelle Geräte und Tablets besitzen jedoch meist einen grösseren Bildschirm. Die Werte und Dateien des „res“-Ordners [48], speziell die Dateien der Ordner „layout“ sowie „values“ und „drawable“ müssen deshalb für grössere Bildschirmgrössen ebenfalls existieren.

Hilfsseite

Des Weiteren könnte eine Hilfsseite eingebaut werden. Dort würde dem Benutzer erklärt, dass keine sensiblen Daten während der Unterhaltung gespeichert werden. Ebenfalls würde dort darauf hingewiesen, dass die Kalibrierung erweitert werden kann. Da die Berechnung mit einer Minute Trainingsdaten nur wenige Sekunden dauert, könnte der Text, welcher während dem Berechnen des Modells angezeigt wird, weggelassen werden. Stattdessen wird dieser Text in der Hilfsseite eingebunden. Während dem Berechnen des Modells würde dem Benutzer dann nur noch der bereits bestehende Fortschrittsbalken angezeigt werden.

Unterhaltung speichern

Momentan wird die Unterhaltung nach dem Beenden nicht gespeichert. Im GUI müsste die Möglichkeit geschaffen werden, dass die Unterhaltung beim Beenden etwa durch einen weiteren Button oder automatisches Speichern gesichert wird. Anschliessend müsste das „TalkStatisticData“-Objekt serialisiert und lokal gespeichert werden. Für die Darstellung einer gespeicherten Unterhaltung kann die „TalkAnalysis“-Activity verwendet werden. Diese wurde so implementiert, dass sie mit „TalkStatisticData“-Objekten arbeitet. Zudem müsste eine Übersichtsseite für die gespeicherten Unterhaltungen erstellt werden.

Lizenzen

In der Applikation wurden zwei Icons [49] [50] verwendet, bei welchen der Designer in der Applikation genannt werden muss [51]. Dies wurde noch nicht realisiert. Die Icons könnten jedoch auch ausgetauscht werden. Betroffen wären die Dateien „magnifying_small.png“ und „return_small.png“ im „res/drawable“-Ordner.

Der Code der RangeSeekBar und die Apache Commons Math-Bibliothek stehen unter der Lizenz „Apache License 2.0“. Diese erfordert, dass eine Kopie der Lizenz in der Software enthalten ist und in der Applikation darauf verwiesen wird [52]. Dies ist noch umzusetzen.

Diagramme

Bei der „TalkAnalysis“-Activity ist kein Wischen zwischen den Bildschirmseiten möglich. Dies wurde ausgeschaltet, da es zu Komplikationen mit der AChartEngine-Bibliothek führt. Um das trotzdem zu ermöglichen, könnte ein eigener ViewPager [53] eingesetzt werden, welcher das Wischen für die View [17] des gestapelten Säulendiagramms deaktiviert, jedoch das sich Fortbewegen im Diagramm erlaubt.

Des Weiteren existiert das Problem, dass nicht immer dieselben Farben pro Profil für die Diagramme verwendet werden. Mit mehreren intensiven Untersuchungen konnte das Problem nicht behoben werden. Zu erwähnen ist, dass auf dem Entwickler-Gerät jeweils die Profile bei den Kreisdiagrammen dieselbe Farbe besitzen, jedoch das gestapelte Säulendiagramm nicht. Auf einem weiteren Gerät [25] werden jeweils für die beiden Diagramme der „TalkAnalysis“-Activity, jedoch nicht für das Diagramm der „Talk“-Activity die gleichen Farben verwendet. Es kommt teilweise auch vor, dass für die Profifarbe in allen Diagrammen dieselbe Farbe verwendet wird. Nach einem Programmneustart kann dies jedoch bereits nicht mehr der Fall sein. Unter diesem Umstand wurden neutrale Profifarben verwendet. Ebenfalls könnte in Erwägung gezogen werden, die Diagramm-Bibliothek „Androidplot“ zu verwenden, da mit AChartEngine diverse Komplikationen existieren bzw. bei bestimmten Komponenten keine feine Einstellung definiert werden kann.

6.2.3 Codestruktur

Es kann sein, dass Methoden für die neuen Komponenten benötigt werden, welche noch nicht bestehen. Mit dem Kapitel 4.2 sollte es jedoch möglich sein, diese in der korrekten Klasse zu implementieren.

6.2.4 Sprechererkennung

Für die Sprechererkennung müssen umfangreiche Parametertests auf eine genügend grosse Menge an Testdaten durchgeführt werden. In diesen Testdaten müssen diverse Alltagssituationen, in welcher ein Einsatz der Applikation als wahrscheinlich betrachtet wird, vorhanden sein. Es gilt unter anderem auszuwerten, wie sich die Distanz des Handys auswirkt, mit Störfaktoren umgegangen werden kann oder die Eigenschaften von verschiedensten Gesprächsteilnehmern bzw. deren Unterhaltungsthema die Leistung des Algorithmus beeinflusst. Mit den erhaltenen Resultaten muss festgestellt werden, ob gute Resultate erreichbar sind bzw. was dafür notwendig ist.

Stille-Filter

Der implementierte Stille-Filter wurde den Umständen der Evaluierung angepasst. Dort kamen verbesserte Resultate zustande, falls die Audiodaten nach Lautstärke gefiltert wurden. Bei einzelnen Tests ergab der Einsatz damit jedoch auch verschlechterte Resultate. Die Ursache konnte nicht gefunden werden. Ebenfalls werden bei einer grösseren Distanz, etwa bei zweieinhalb Meter, das Mikrophon entgegen der Richtung des Sprechers ausgerichtet, die Unterschiede bezüglich Lautstärke von Stille und Stimme sehr gering. Dies wurde mit Visualisierungen in Matlab ausgewertet. Deshalb wird die Implementation davon als nicht abschliessend betrachtet. Es wäre genau zu überprüfen, unter welchen Umständen verschlechterte Resultate erhalten werden bzw. wie die Parameter zu setzen sind.

Schwellenwert

Eine automatische Bestimmung des Schwellenwertes wäre wünschenswert, da die resultierenden Log-Likelihood-Werte pro Sprechermodell jeweils abweichen. Der in der Evaluierung gezeigte Ansatz könnte diesbezüglich Verbesserungen bringen. Zu bedenken ist, dass mit den aktuell erzeugten Sprechermodellen keine konstant guten Ergebnisse resultieren. Deshalb müsste dieser Ansatz nach einer Verbesserung bezüglich den Sprechermodellen erneut überprüft werden. Kann ein Perzentil definiert werden, so müsste für eine benutzerfreundliche Applikation eine möglichst geringe Menge für die benötigten Testdaten ausgemacht werden. Die Menge bzw. Anzahl MFCCs würde vor dem Erstellen des Modells von den Trainingsdaten separiert. Ist das Modell erstellt worden, so würden die nicht verwendeten MFCCs verwendet, um die Log-Likelihood-Werte für das Modell zu erhalten.

Verschiedene Modelle

Ebenfalls wären die Unterschiede zwischen mehreren, mit den gleichen Daten, erzeugten Modellen zu untersuchen. Würden dort grosse Abweichungen auftreten, so müsste auch hier eine Lösung geschaffen werden. Ein möglicher Ansatz wäre, mit einem grossen Teil der Trainingsdaten mehrere Modelle zu erzeugen und anschliessend mit dem kleinen Teil der Trainingsdaten zu testen. Zusätzlich müsste damit ein weiteres Modell getestet werden, welches nicht von dieser Person ist. Hier würde der Einsatz eines UBM (siehe Kapitel 6.3.1) auch Sinn machen. Anschliessend würde jenes erzeugte Modell verwendet,

welches die grössten Abweichungen bei den erhaltenen Log-Likelihood-Werten vom eigenen und dem zusätzlichen Modell besitzt. Es ist jedoch fraglich, ob sich dieser Aufwand lohnt, und wie dies benutzerfreundlich durchgeführt werden könnte. Eine Möglichkeit wäre, direkt nach der Kalibrierung das erstberechnete Modell zu verwenden und anschliessend mit einem IntentService [54] die weiteren Schritte nachträglich im Hintergrund durchzuführen.

Energieverbrauch

Der Energieverbrauch ist ein wesentlicher Faktor auf mobilen Geräten. Es wäre deshalb nötig, den Energieverbrauch der Applikation zu messen. Würde die Applikation zu viel Strom benötigen, müssten Komponenten eingesetzt oder Parameter geändert werden, mit welchen der Stromverbrauch gesenkt werden kann.

Externe Rechenleistung

Weil das Berechnen des Sprechermodells einige Zeit dauert, könnte dies extern an einen Serverdienst ausgelagert werden. Dieser empfängt MFCCs und erstellt daraus ein Sprechermodell, welches er dem Handybenutzer zurückschickt. Dadurch würde jedoch Internetverkehr entstehen. Da die MFCCs persönliche Daten enthalten wird eine verschlüsselte Kommunikation empfohlen. Des Weiteren ist zu bedenken, dass eine gewisse Übertragungszeit entsteht und Datenvolumen verursacht wird.

6.3 Strategischer Ausblick

Als langfristiges Ziel kann betrachtet werden, eine geeignete Methode bzw. Komponente für die Sprechererkennung, welche auch unter realen Bedingungen konstant gute Resultate erbringt, zu evaluieren. Dies wird jedoch als aufwändige und komplexe Arbeit betrachtet.

6.3.1 Sprechererkennung

Nachfolgend werden zwei Ansätze für verbesserte Resultate betreffend Sprechererkennung erwähnt, welche genauer zu prüfen wären.

Ansatz von der Applikation MyConverse [24]

Die Resultate der Sprechererkennung ergaben in der Evaluation keine befriedigenden Resultate. Ebenfalls wird vom Benutzer Zeit für die Kalibrierung vorausgesetzt. Die Applikation MyConverse verkürzt diese durch den Einsatz eines „Universal Background Models“ (UBM). Dies ist ein GMM, welches von Stimmen diverser Personen erstellt wurde. Mit MFCCDDs, das sind MFCCs mit Delta- und Delta-Delta-Features, und einem GMM-UBM wurde eine bemerkbar höhere Genauigkeit als mit MFCCs und einem GMM erreicht. Dieser Ansatz könnte deshalb durchaus in Betracht gezogen werden.

Ein UBM könnte in der Talkalyzer-Applikation dem Sprecherprofil „Andere“ zugeordnet werden. Zudem müsste eine neue „DecisionMaker“-Klasse geschrieben werden, welche das Wahrscheinlichkeitsverhältnis zwischen dem Modell des Benutzerprofils und dem UBM durchführt.

Parallel ausgeführte Bachelorarbeit

Ebenfalls wären die Resultate der parallel ausgeführten Bachelorarbeit (siehe Kapitel 8.1.2) zu prüfen. Konnte damit ein verbesserter Ansatz evaluiert werden und die Ressourcenanforderungen für ein Handy würden nicht übertroffen, so wäre dies als ausgezeichnete Erweiterung in Erwägung zu ziehen.

7.1 **Literaturverzeichnis**

- [1] Unperfekthaus, „Talk-o-Meter on the App Store on iTunes“ Unperfekthaus, Jahr unbekannt. [Online]. Available: <https://itunes.apple.com/us/app/talk-o-meter/id341055818?mt=8>. [Zugriff am 23 05 2015].
- [2] M. Brookes, „VOICEBOX“ Imperial College London, Jahr unbekannt. [Online]. Available: <http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html>. [Zugriff am 23 05 2014].
- [3] M. Schedl, „CoMIRVA“ Markus Schedl, 19 12 2011. [Online]. Available: <http://www.cp.jku.at/people/schedl/Research/Development/CoMIRVA/webpage/CoMIRVA.html>. [Zugriff am 16 05 2014].
- [4] Google, „Getting Started with Android Studio | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/sdk/installing/studio.html>. [Zugriff am 28 05 2014].
- [5] Google, „Processes and Threads | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/guide/components/processes-and-threads.html>. [Zugriff am 16 05 2014].
- [6] Google, „Activities | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/guide/components/activities.html>. [Zugriff am 26 02 2014].
- [7] Google, „Bound Services | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/guide/components/bound-services.html>. [Zugriff am 28 05 2014].
- [8] Google, „Services | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/guide/components/services.html>. [Zugriff am 19 03 2014].
- [9] Autor unbekannt, „k-Means-Algorithmus – Wikipedia“ Wikimedia Foundation, 16 04 2014. [Online]. Available: <http://de.wikipedia.org/wiki/K-Means-Algorithmus#k-Means.2B.2B>. [Zugriff am 16 05 2014].
- [10] M. Roughan, „File Exchange - MATLAB Central“ The MathWorks, Inc., 28 10 2009. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/24867-gaussian-mixture-model-m/content/gaussian_mixture_model.m. [Zugriff am 16 05 2014].
- [11] Autor unbekannt, „Maximum-Likelihood-Methode – Wikipedia“ Wikimedia Foundation, 03 05 2014. [Online]. Available: <http://de.wikipedia.org/wiki/Maximum-Likelihood-Methode>. [Zugriff am 16 05 2014].
- [12] Google, „<uses-sdk> | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels>. [Zugriff am 26 02 2014].
- [13] A. Haider, „Man in suit Free Stock Photo - Free Images“ Stockvault.net, 18 01 2010. [Online]. Available: <http://www.stockvault.net/photo/112807/man-in-suit>. [Zugriff am 05 31 2014].
- [14] greyerbaby, „emily and xemenia 139.JPG“ morgueFile, 07 11 2012. [Online]. Available: <http://www.morguefile.com/archive/display/841114>. [Zugriff am 31 05 2014].
- [15] Google, „Android NDK | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <https://developer.android.com/tools/sdk/ndk/index.html>. [Zugriff am 27 03 2014].
- [16] T. Stadelmann, „Voice Modeling Methods for Automatic Speaker Recognition“ 15 04 2010. [Online]. Available: <http://archiv.ub.uni-marburg.de/diss/z2010/0465/view.html>. [Zugriff am 23 05 2014].
- [17] Google, „View | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/reference/android/view/View.html>. [Zugriff am 25 04 2014].

- [18] AppBrain, „AndroidPlot - Android library statistics | AppBrain.com“ AppBrain, Jahr unbekannt. [Online]. Available: <http://www.appbrain.com/stats/libraries/details/androidplot>. [Zugriff am 19 03 2014].
- [19] AppBrain, „AChartEngine - Android library statistics | AppBrain.com“ AppBrain, Jahr unbekannt. [Online]. Available: <http://www.appbrain.com/stats/libraries/details/achartengine>. [Zugriff am 19 03 2014].
- [20] Carnegie Mellon University, „pocketsphinx on Android - CMUSphinx Wiki“ Carnegie Mellon University, 05 02 2014. [Online]. Available: [http://cmusphinx.sourceforge.net/wiki/tutorialandroid?s\[\]=android](http://cmusphinx.sourceforge.net/wiki/tutorialandroid?s[]=android). [Zugriff am 10 05 2014].
- [21] dhdfu, „cmusphinx/PocketSphinxAndroidDemo at trunk · cjac/cmusphinx · GitHub“ Produktionsfirma unbekannt, 2010. [Online]. Available: <https://github.com/cjac/cmusphinx/tree/trunk/PocketSphinxAndroidDemo/src/edu/cmu/pocketsphinx/demo>. [Zugriff am 10 05 2014].
- [22] The Apache Software Foundation, „Apache Ant - Welcome“ The Apache Software Foundation, 26 05 2014. [Online]. Available: <http://ant.apache.org/>. [Zugriff am 29 05 2014].
- [23] D. Beazley, „Simplified Wrapper and Interface Generator“ SWIG, 27 05 2014. [Online]. Available: <http://www.swig.org/>. [Zugriff am 29 05 2014].
- [24] M. Rossi, O. Amft, S. Feese, C. Käslin und G. Tröster, „UbiComp“ 8 09 2013. [Online]. Available: <http://www.ubicomp.org/ubicomp2013/adjunct/adjunct/p1275.pdf>. [Zugriff am 26 05 2014].
- [25] Google, „Nexus 4 - Google“ Google, Jahr unbekannt. [Online]. Available: <http://www.google.com/intl/ALL/nexus/4/>. [Zugriff am 01 06 2014].
- [26] Morguefile, „morgueFile free photos“ Morguefile, Jahr unbekannt. [Online]. Available: <http://morguefile.com/about>. [Zugriff am 01 05 2014].
- [27] Google, „Toasts | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/guide/topics/ui/notifiers/toasts.html>. [Zugriff am 03 06 2014].
- [28] Google, „SeekBar | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/reference/android/widget/SeekBar.html>. [Zugriff am 11 05 2014].
- [29] tittel@kom.e-technik.tu-darmstadt.de, „range-seeking-bar - A slider widget for Android allowing to set a minimum and maximum value on a numerical range. - Google Project Hosting“ Google Project Hosting, Jahr unbekannt. [Online]. Available: <http://code.google.com/p/range-seeking-bar/>. [Zugriff am 11 05 2014].
- [30] Autor unbekannt, „Index of /snapshot/org/achartengine/achartengine/1.2.0“ achartengine, 23 05 2014. [Online]. Available: <https://repository-achartengine.forge.cloudbees.com/snapshot/org/achartengine/achartengine/1.2.0/>. [Zugriff am 23 05 2014].
- [31] Google, „Fragments | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/guide/components/fragments.html>. [Zugriff am 26 02 2014].
- [32] Google, „AudioRecord | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/reference/android/media/AudioRecord.html>. [Zugriff am 21 04 2014].
- [33] Google, „Parcel | Android Developers“ Google, 30 05 2014. [Online]. Available: <http://developer.android.com/reference/android/os/Parcel.html>. [Zugriff am 02 06 2014].
- [34] Google, „Intent | Android Developers“ Google, 30 05 2014. [Online]. Available: <http://developer.android.com/reference/android/content/Intent.html>. [Zugriff am 02 06 2014].
- [35] Google, „Gson - Google Code“ Google, Jahr unbekannt. [Online]. Available: <https://code.google.com/p/google-gson/>. [Zugriff am 18 04 2014].
- [36] The Apache Software Foundation, „Math - Commons Math: The Apache Commons Mathematics Library“ The Apache Software Foundation, 03 11 2013. [Online]. Available: <http://commons.apache.org/proper/commons-math/>. [Zugriff am 11 05 2014].

- [37] Google, „Storage Options | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/guide/topics/data/data-storage.html#pref>. [Zugriff am 18 04 2014].
- [38] Google, „AsyncTask | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/reference/android/os/AsyncTask.html>. [Zugriff am 12 05 2014].
- [39] Google, „TimerTask | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/reference/java/util/TimerTask.html>. [Zugriff am 12 05 2014].
- [40] OrangeCow, „oc volume - java speech recognition engine“ OrangeCow organization, Jahr unbekannt. [Online]. Available: <http://ocvolume.sourceforge.net/>. [Zugriff am 21 04 2014].
- [41] Google, „Serializable | Android Developers“ Google, 13 05 2014. [Online]. Available: <http://developer.android.com/reference/java/io/Serializable.html>. [Zugriff am 17 05 2014].
- [42] Google, „singlesound - Simple raw-wav audio I/O and processing helper tool for java. - Google Project Hosting“ Google Project Hosting, Jahr unbekannt. [Online]. Available: <https://code.google.com/p/singlesound/>. [Zugriff am 01 05 2014].
- [43] Autor unbekannt, „Fourier transform - Wikipedia, the free encyclopedia“ Wikimedia Foundation, 18 05 2014. [Online]. Available: http://en.wikipedia.org/wiki/Fourier_transform. [Zugriff am 25 05 2014].
- [44] Autor unbekannt, „Sampling (signal processing) - Wikipedia, the free encyclopedia“ Wikimedia Foundation, 07 05 2014. [Online]. Available: [http://en.wikipedia.org/wiki/Sampling_\(signal_processing\)](http://en.wikipedia.org/wiki/Sampling_(signal_processing)). [Zugriff am 05 30 2014].
- [45] M. Brookes, „Description of melcepst“ Imperial College London, 23 04 2014. [Online]. Available: <http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/doc/voicebox/melcepst.html>. [Zugriff am 16 05 2014].
- [46] Autor unbekannt, „F1 score - Wikipedia, the free encyclopedia“ Wikimedia Foundation, 17 05 2014. [Online]. Available: http://en.wikipedia.org/wiki/F1_score. [Zugriff am 21 05 2014].
- [47] D. Mazzone und R. Dannenberg, „Audacity: Freier Audioeditor und Rekorder“ The Audacity Team, Jahr unbekannt. [Online]. Available: <http://audacity.sourceforge.net/>. [Zugriff am 26 05 2014].
- [48] Google, „Providing Resources | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/guide/topics/resources/providing-resources.html>. [Zugriff am 26 05 2014].
- [49] Freepik, „Magnifying glass on pie chart vector icon | Free Business icons“ FLATICON, Jahr unbekannt. [Online]. Available: http://www.flaticon.com/free-icon/return_3998. [Zugriff am 01 06 2014].
- [50] A. Whitcroft, „Return vector icon | Free Arrows icons“ FLATICON, Jahr unbekannt. [Online]. Available: http://www.flaticon.com/free-icon/return_3998. [Zugriff am 01 06 2014].
- [51] Creative Commons, „Creative Commons — Attribution 3.0 Unported“ Creative Commons, Jahr unbekannt. [Online]. Available: <http://creativecommons.org/licenses/by/3.0/>. [Zugriff am 01 06 2014].
- [52] The Apache Software Foundation, „Apache License and Distribution FAQ“ The Apache Software Foundation, Jahr unbekannt. [Online]. Available: <http://www.apache.org/foundation/license-faq.html#WhatDoesItMEAN>. [Zugriff am 02 06 2014].
- [53] Google, „ViewPager | Android Developers“ Google, 13 05 2014. [Online]. Available: <http://developer.android.com/reference/android/support/v4/view/ViewPager.html>. [Zugriff am 19 05 2014].
- [54] Google, „IntentService | Android Developers“ Google, 30 05 2014. [Online]. Available: <http://developer.android.com/reference/android/app/IntentService.html>. [Zugriff am 01 06 2014].
- [55] D. Reynolds, „Gaussian Mixture Models“ MIT Lincoln Laboratory, Lexington, MA 02421, USA, Jahr unbekannt.

- [56] Autor unbekannt, „Apple iOS – Wikipedia“ Wikimedia Foundation, 16 05 2014. [Online]. Available: http://de.wikipedia.org/wiki/Apple_iOS. [Zugriff am 23 05 2014].
- [57] Google, „Layout Resource | Android Developers“ Google, Jahr unbekannt. [Online]. Available: <http://developer.android.com/guide/topics/resources/layout-resource.html>. [Zugriff am 26 05 2014].
- [58] The Apache Software Foundation, „MixtureMultivariateNormalDistribution (Apache Commons Math 3.3 API)“ The Apache Software Foundation, 15 05 2014. [Online]. Available: <http://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/distribution/MixtureMultivariateNormalDistribution.html>. [Zugriff am 02 06 2014].

7.2 Abbildungsverzeichnis

Abbildung 1 - Visualisierung für die Zuordnung von Elementen in eine Gruppe anhand von Schwerpunkten	10
Abbildung 2 - Gaussian-Mixture-Model (Bildquelle: [10])	12
Abbildung 3 - Persona Felix Burkhalter (Bildquelle: [13])	16
Abbildung 4 - Persona Selina Meier (Bildquelle: [14])	16
Abbildung 5 - Gestapeltes Säulendiagramm	19
Abbildung 6 - Bewertung der Diagramm-Bibliotheken	22
Abbildung 7 – Bewertung der Sprechererkennungs-Bibliotheken	23
Abbildung 8 – Vorarbeit für Kalibrierung in der Calibration-Activity	26
Abbildung 9 - Bilder beschreiben in der Calibration-Activity	27
Abbildung 10 – Kalibrierung abschliessen in der Calibration-Activity	28
Abbildung 11 – Launch-Activity	28
Abbildung 12 – Talk-Activity	29
Abbildung 13 – RangeSeekBar	29
Abbildung 14 - TalkAnalysis-Activity	30
Abbildung 15 - UML-Diagramm der Klasse Talk	37
Abbildung 16 - UML-Diagramm der Klasse TalkStatisticData	37
Abbildung 17 - Sprecherwechsel bei der Audiodatei 1	54
Abbildung 18 - Sprecherwechsel bei der Audiodatei 2	56

7.3 Diagramme

Diagramm 1 - Ablauf der Applikation	7
Diagramm 2 - Ablauf Sprechererkennung	10
Diagramm 3 - Ablauf Modell generieren	20
Diagramm 4 - Ablauf Sprechererkennung	20
Diagramm 5 - Sprecherbestimmung	20
Diagramm 6 - Kalibrierung erweitern	21
Diagramm 7 – Übersicht austauschbare Komponenten	31
Diagramm 8 - UML-Diagramm für die Schnittstelle IFeatureConverter	32
Diagramm 9 - UML-Diagramm für die Klasse AndroidMicrophoneInputStream	32
Diagramm 10 - UML-Diagramm der implementierten Feature Extraktion	33
Diagramm 11 - UML-Diagramm für die Schnittstelle IConvertedAudioData	33
Diagramm 12 - UML-Diagramm für die Klasse MfccData	33
Diagramm 13 - UML-Diagramm der Klasse MfccSilenceFilter und der Schnittstelle IFilterSilence ..	33
Diagramm 14 - UML-Diagramm Audiodaten einlesen und verarbeiten	34
Diagramm 15 - UML-Diagramm der Schnittstelle „IStateObject“	34

Diagramm 16 - UML-Diagramm der Klasse GaussianMixtureModel und der Schnittstelle IDefaultTalkerModel.....	35
Diagramm 17 - UML-Diagramm der Klasse Calibration.....	35
Diagramm 18 - UML Diagramm für die Klasse ModelCreatorRunnable.....	36
Diagramm 19 - UML-Diagramm Kalibrierung.....	36
Diagramm 20 - UML-Diagramm für die Klasse DecisionMaker und die Schnittstelle IDecisionMaker	37
Diagramm 21 - UML-Diagramm für die Klasse AudioAnalysisRunnable.....	38
Diagramm 22 - UML-Diagramm der Klasse TalkThreadManager und der Schnittstelle IThreadManager.....	38
Diagramm 23 - UML-Diagramm Unterhaltung führen.....	38
Diagramm 24 - Ablauf Kalibrierung.....	41
Diagramm 25 - Ablauf Unterhaltung führen.....	42
Diagramm 26 - Ablauf Kalibrierung erweitern.....	43
Diagramm 27 – Vorgehen um die MFCCs zu evaluieren.....	49
Diagramm 28 – Vorgehen um k-Means- und Sprechermodell-Implementation in der Talkalyzer- Applikation auf Android zu evaluieren.....	50
Diagramm 29 – Likelihood-Kurven für das Vorgehen aus Tabelle 27.....	51
Diagramm 30 – Likelihood-Kurven für zwei verschiedene Modelle generiert aus Voicebox-MFCCs in Matlab.....	51
Diagramm 31 - Likelihood-Kurven für das Vorgehen aus Tabelle 28.....	52
Diagramm 32 - Likelihood-Kurven für zwei verschiedene Modelle generiert aus CoMIRVA-MFCCs auf Android.....	52
Diagramm 33 – Schwellenwert für ein Sprechermodell bestimmen.....	54
Diagramm 34 – Log-Likelihood-Werte von Person 1 (links) und Person 4 (rechts).....	57

7.4 Tabellen

Tabelle 1 – Terminologie.....	8
Tabelle 2 - Legende für Tabelle 3.....	11
Tabelle 3 – Benötigte Berechnungen anhand dem Resultat des k-Means-Algorithmus.....	11
Tabelle 4 – Applikationseigenschaften.....	13
Tabelle 5 - Hauptaktivitäten der Applikation.....	13
Tabelle 6 - Anwendungsfall 1.01.....	14
Tabelle 7 - Anwendungsfall 1.02.....	15
Tabelle 8 - Anwendungsfall 1.03.....	15
Tabelle 9 - Kriterien für das GUI.....	17
Tabelle 10 – Untersuchte Kalibrierungsmethoden.....	18
Tabelle 11 - Faktoren für die Diagramme.....	19
Tabelle 12 - Austauschbare Komponenten.....	21
Tabelle 13 - Implementierte Activities.....	26
Tabelle 14 - Beschreibung der Klasse AudioStreamConverterRunnable und AndroidMicrophoneInputStream sowie der Schnittstelle IFeatureConverter.....	32
Tabelle 15 - Beschreibung der Klassen MfccData und MfccSilenceFilter sowie den Schnittstellen IConvertedAudioData und IFilterSilence.....	34
Tabelle 16 - Beschreibung der Schnittstelle IStateObject.....	34
Tabelle 17 - Beschreibung der Klasse GaussianMixtureModel und der Schnittstelle IDefaultTalkerModel.....	35
Tabelle 18 -Beschreibung der Klassen Calibration, ModelCreatorRunnable und CalibrationService..	36
Tabelle 19 - Beschreibung der Klasse DecisionMaker und der Schnittstelle IDecisionMaker.....	37
Tabelle 20 - Beschreibung der Klassen Talk, TalkStatisticData, TalkService und AudioAnalysisRunnable.....	38
Tabelle 21 - Beschreibung der Klassen AudioParameters, ModelParameters und MfccLibrarySpecificParameters.....	39

Tabelle 22 - Zugehörigkeiten und Abhängigkeiten der austauschbaren Komponenten.....	39
Tabelle 23 - Komponenten des Sprechererkennungsalgorithmus und dessen Implementation	40
Tabelle 24 - Personen für die Evaluation des GUIs	44
Tabelle 25 - Beschreibung Austauschverfahren für Audio-Merkmale extrahieren.....	45
Tabelle 26 - Beschreibung Austauschverfahren für das Sprechererkennungsmodell für die Kalibrierung.....	46
Tabelle 27 - Beschreibung Austauschverfahren für die Entscheidungslogik der Sprecherbestimmung	47
Tabelle 28 - Beschreibung Austauschverfahren für den Filter von Audiodaten	47
Tabelle 29 - Parameter in der Applikation auf Android für die Evaluierung	48
Tabelle 30 - Parameter in der Applikation in Matlab für die Evaluierung.....	49
Tabelle 31 - Vorgehen um die MFCCs zu evaluieren	49
Tabelle 32 - Vorgehen um k-Means- und Sprechermodell-Implementation in der Talkalyzer- Applikation auf Android zu evaluieren	50
Tabelle 33 - Legende für das F-Mass	53
Tabelle 34 - Beschreibung des Aufbaus der Audiodatei 1	54
Tabelle 35 - Resultate für verschiedene Trainingszeiten bei Gruppe 1.....	55
Tabelle 36 – Resultate für verschiedene Trainingszeiten bei Gruppe 2	55
Tabelle 37 - Resultate für verschiedene Trainingszeiten bei Gruppe 3.....	55
Tabelle 38 - Durchschnittswerte für das F-Mass mit verschiedenen Trainingszeiten.....	56
Tabelle 39 - Beschreibung des Aufbaus der Audiodatei 2	56
Tabelle 40 - Resultate für die Audiodatei 2.....	57
Tabelle 41 - Werte für das F-Mass für diverse Perzentile	58
Tabelle 42 - F-Mass bei fünf Personen pro Gruppe und gegen den Sprecher gerichtetes Mikrofon	58

8.1 Projektmanagement

8.1.1 Offizielle Aufgabenstellung

Zürcher Hochschule
für Angewandte Wissenschaften



**School of
Engineering**

Talkalyzer: Mobile-App zur automatischen Sprecher- Erkennung BA14_ciel_1

BetreuerInnen: Mark Cieliebak, ciel
Thilo Stadelmann, stdm
Fachgebiete: Software (SOW)
Studiengang: IT
Zuordnung: Institut für angewandte Informationstechnologie (InIT)
Gruppengrösse: 1

Kurzbeschreibung:

In dieser Bachelor-Arbeit (BA) soll eine Mobile-App entwickelt werden, die für eine Diskussion oder Besprechung live ermittelt, wer wie viel redet. Damit soll es z.B. einem Vorgesetzten möglich sein zu erkennen, ob er in einem Mitarbeitergespräch zu viel redet oder ob das Gespräch ausgeglichen ist. Analog könnte auch ein Ehepaar feststellen, wer in den Diskussionen das Sagen hat.

Hintergrund: Am InIT wurde in einem Forschungsprojekt bereits ein Prototyp entwickelt, der die Besonderheiten der Stimme eines Menschen aus Tonaufnahmen extrahiert. Das System ist anschliessend in der Lage, für neue Tonaufnahmen zu erkennen, wann der zuvor "trainierte" Sprecher redet. Dies funktioniert auch, wenn mehrere Personen miteinander sprechen. Der Code wurde in Matlab und Java implementiert und ist sehr kompakt und übersichtlich.

Aufgabe: Im Rahmen dieser BA entwickeln Sie eine Mobile-App, mit der Benutzer live erkennen können, wer wann und wie viel spricht. Dazu soll der vorhandene Code auf Android portiert und rundherum eine ansprechende App gebaut werden. Das Ziel ist eine marktreife Applikation, die Sie im App-Store publizieren können.

Hier einige wesentliche Teilaufgaben, die Sie im Laufe dieser BA bearbeiten werden:

- Entwicklung einer intuitiven Visualisierung der Sprecher-Anteile
- Entwurf einer Architektur, die es erlaubt neue Algorithmen zu Sprechererkennung einfach zu integrieren
- Portierung der existierenden Verfahren/Libraries auf die Mobile Plattform
- Performance-Optimierung, um Analysen in Quasi-Echtzeit (live) zu ermöglichen
- Evaluation, wie gut die Analysen in der Praxis funktionieren.

Parallel zu dieser Arbeit findet eine weitere BA statt, in der die Verfahren zur Sprechererkennung weiterentwickelt und optimiert werden. Beide Arbeiten werden gemeinsam betreut und sollen in Abstimmung realisiert werden. Insbesondere soll es möglich sein, die neuen/verbesserten Verfahren einfach in der App zu integrieren.

Voraussetzungen:

- Freude an intuitiven GUIs und gutem Design
- Entwicklung von Mobile Apps auf Android
- Hilfreich, aber nicht notwendig: Erfahrung mit Matlab

8.1.2 Aufgabenstellung Bachelorarbeit Stampfli

Zürcher Hochschule
für Angewandte Wissenschaften



School of
Engineering

Talkalyzer: Neue Algorithmen für automatische Sprecher- Erkennung BA14_stdm_1

BetreuerInnen: Thilo Stadelmann, stdm
Mark Cieliebak, ciel
Fachgebiete: Datenanalyse (DA)
Digitale Signalverarbeitung (DSV)
Information Security (IS)
Software (SOW)
Studiengang: ET / IT
Zuordnung: Institut für angewandte Informationstechnologie (InIT)
Gruppengrösse: 1

Kurzbeschreibung:

Die computergestützte Erkennung von Personen anhand ihrer Stimme hat viele spannende Anwendungen, z.B.:

- In biometrischen Sicherheitssystemen (etwa zur Zutrittskontrolle in Kraftwerken)
- In Verfahren zur automatischen semantischen Analyse von Audio- und Videorecordings (etwa, um in Spielfilmen nach Szenen mit bestimmten Schauspielern zu suchen)

Am InIT wurde daher ein Prototyp (in Matlab und Java) entwickelt, der die Besonderheiten der Stimme eines Menschen aus Beispielen extrahiert und danach in der Lage ist, für weiteren Sprachinput zu entscheiden, ob dieser von dem zuvor "trainierten" Sprecher stammt oder nicht. Der bestehende Code ist sehr kompakt und übersichtlich. Er implementiert die wesentlichen Stufen der Stimmerkennung und kann in dieser Bachelor-Arbeit (BA) als Beispiel und/oder Grundlage dienen.

Im Rahmen dieser BA sollen nun neue Verfahren implementiert und ausprobiert werden, um die Erkennungsrate von Sprechern zu steigern: Wie kann man beispielsweise die zeitliche Abfolge von Lauten besser für die Identifikation der Stimme nutzen? Hierzu existieren Ansätze in der Literatur und bei den Betreuern, aber auch eigene Ideen dürfen gerne eingebracht werden.

Diese BA richtet sich an Studierende, die sich gerne der Herausforderung einer (sehr praxisrelevanten) Forschungsfrage stellen möchten. Dabei werden keinerlei Vorkenntnisse in Sprachverarbeitung oder Datenanalyse vorausgesetzt. Vielmehr werden Sie durch ein erfahrenes Team und gute Materialien schnell up-to-date gebracht und starten dann Ihre eigenen Untersuchungen. Dabei bietet sich Ihnen die Chance auf eine wirklich neue Entwicklung!

Parallel zu dieser Arbeit findet eine weitere BA statt, in der eine Android-App für Sprechererkennung entwickelt wird. Beide Arbeiten werden gemeinsam betreut und sollen in Abstimmung realisiert werden. Insbesondere soll es möglich sein, die neuen/verbesserten Verfahren einfach in der App zu integrieren.

Voraussetzungen:

- Freude am Programmieren
- Affinität zu algorithmischen Fragestellungen
- Lust, sich vielleicht auch erstmals mit wissenschaftlichen Publikationen zu beschäftigen
- Hilfreich, aber nicht notwendig können sein: Erste Erfahrungen mit Matlab oder C++ (für Lesen von Beispielcode)

Weiterführende Informationen:

https://dublin.zhaw.ch/~stdm/?page_id=77

8.2 Weiteres

8.2.1 Auszug aus dem Mailverkehr über den Ersatz der AudioInputStream-Klasse mit Herr Klaus Seyerlehner

„Also wenn der InputStream PCM Audiodaten für einen Kanal mit der angegebenen SampleRate enthält könnte das mit dem InputStream so funktionieren. Schlußendlich wird ja im Konstruktor nur eine Konvertierung auf das intern erwartete PCM Format durchgeführt.“

8.2.2 Auszug aus dem Mailverkehr bezüglich Copyright [10]

„From my point of view you are welcome as long as it is acknowledged, and as long as it is OK with your thesis advisor -- I don't want you to get into trouble with your University.“