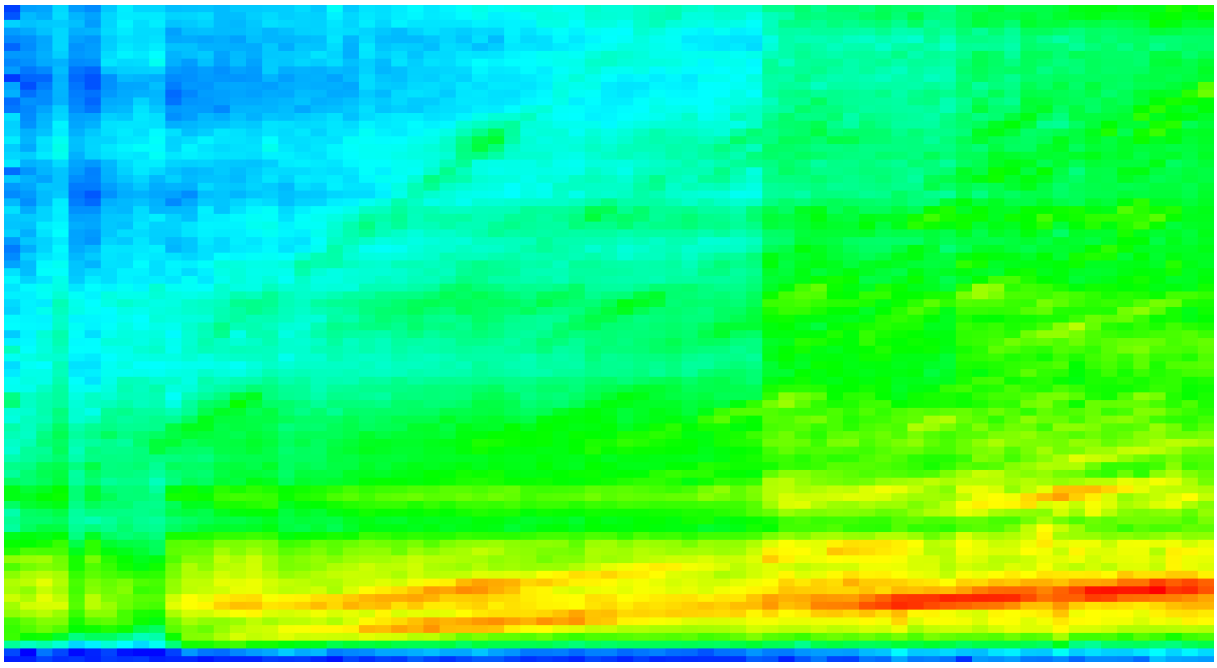
Hector Seminar

IPEK
Institut für Produktentwicklung
am Karlsruher Institut für Technologie

Entwicklung einer NVH-Applikation für mobile Endgeräte



Abschlussbericht der Kooperationsphase 2019/20

Durchgeführt am Institut für Produktentwicklung am
Karlsruher Institut für Technologie
Betreut durch Manuel Bopp und Sebastian Lutz

Aiman Malek, Yannik Tausch

Inhaltsverzeichnis

Abstract

During the development of today's automobile, the discipline of automotive engineering has evolved continuously. At the same time, aspects beyond the bare functionality of the car like passenger comfort become more and more important. This also includes good sound design – especially the reduction of unwanted **noise, vibration, and harshness (NVH**). To measure these effects during the prototyping phase, specifically designed test benches with industrial microphones are heavily utilized. Besides the audio recording, the rotation speed of the engine must be logged. While being appropriate for high precision requirements in specific stages of development, lower precision measurements can often be accepted. This opens the possibility to use different measurement approaches like applications for mobile devices. Mobile apps combine cost effectiveness and the ability to take measurements nearly everywhere. Such apps already exist - but they are not freely available and under a proprietary license. Consequently, the objective of the project is to **develop an application for Android mobile devices accessible to anyone that can convey a basic acoustic analysis without the need to carry cost-intensive equipment**.

With this solution, the built-in microphone of the smartphone or tablet records the audio data while an inexpensive Bluetooth adapter connected to the car's OBD diagnostic port gathers the required engine speed data. The data is then analysed and visualized locally on the device.

1 Einführung

In der Fahrzeugtechnik achten Ingenieure bei der Konstruktion eines Kraftfahrzeugs neben der Fahrtauglichkeit auch auf den Fahrkomfort. Darunter fallen auch Eigenschaften im Bereich **Noise Vibration Harshness** (**NVH**, „Geräusch, Vibration, Rauigkeit“). NVH umfasst allgemein die hör- oder spürbaren Schwingungen in Kraftfahrzeugen und Maschinen.

Die allgemeine Ursache für Schwingungen sind periodische Kraftanregungen. Ein Beispiel dafür sind sogenannte Stick-Slip-Effekte, auch Haftgleiteffekte genannt, welche durch ruckartiges Gleiten von gegeneinander bewegten Festkörpern zustande kommen. Solche Reibungen führen zur Abstrahlung von Körperschall, also einem sich im Festkörper ausbreitenden Schall, oder hörbarem Luftschall. Beispiele für NVH sind quietschende Bremsen, Getriebeheulen oder ratternde Scheibenwischer.

NVH wird an Akustikprüfständen gemessen. Diese sind häufig anechoische Kammern, also für Schall reflexionsarme Räume. Für die Messungen werden industrielle feinfühlige Mikrofone und Beschleunigungssensoren verwendet. Bei Störungen in der Akustik des Motors eines Kraftfahrzeugs (z.B. Motorheulen) wird ein Zusammenhang zwischen störendem Geräusch und der Motordrehzahl angenommen, weshalb diese zusätzlich gemessen wird. Die Messergebnisse werden danach von einer Auswertungssoftware visualisiert.

Eine häufig verwendete Visualisierungsform ist das **Campbell-Diagramm** (siehe Abb. 1.1). Dieses setzt die Drehzahl eines Rotors in Relation mit den zur Schwingung angeregten Frequenzen. Wirkt eine breitbandige Anregung auf ein System, so ist diejenige Frequenz, welche mit der größten Amplitude schwingt, die **Resonanzfrequenz**. Im NVH-Bereich besteht häufig ein **linearer Zusammenhang** zwischen der Drehzahl des Motors und der Frequenz der zur Schwingung angeregten Bauteile.

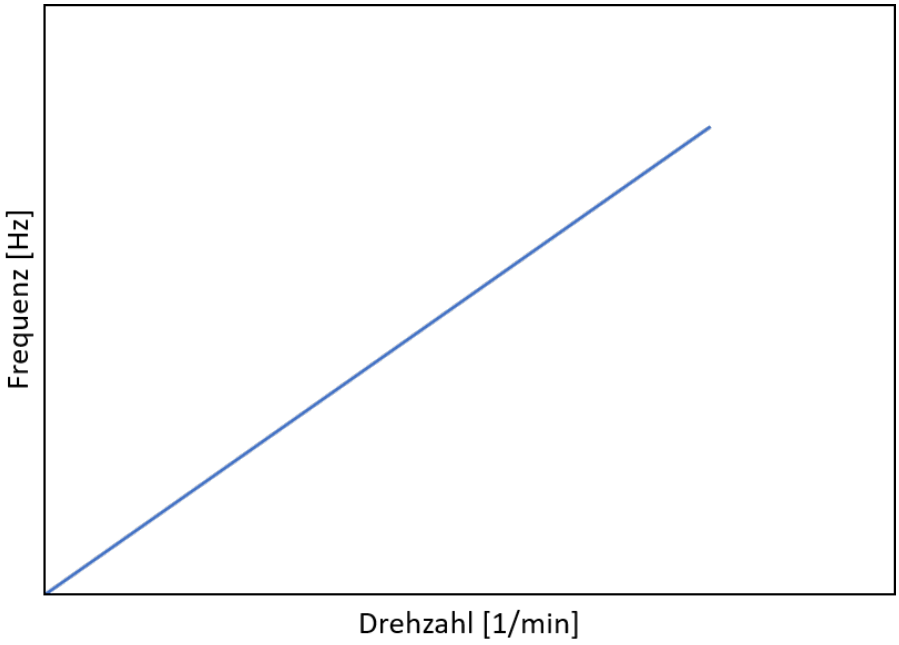


Abbildung 1.1: Einfaches Beispiel eines Campbell-Diagrammes

Um das Diagramm um eine dritte Dimension, welche die Amplitude der jeweiligen Frequenz darstellt, zu erweitern, verwendet man **Campbell-Spektrogramme**, welche die Amplitude durch einen Farbgradienten kennzeichnen (siehe Abb. 1.2).

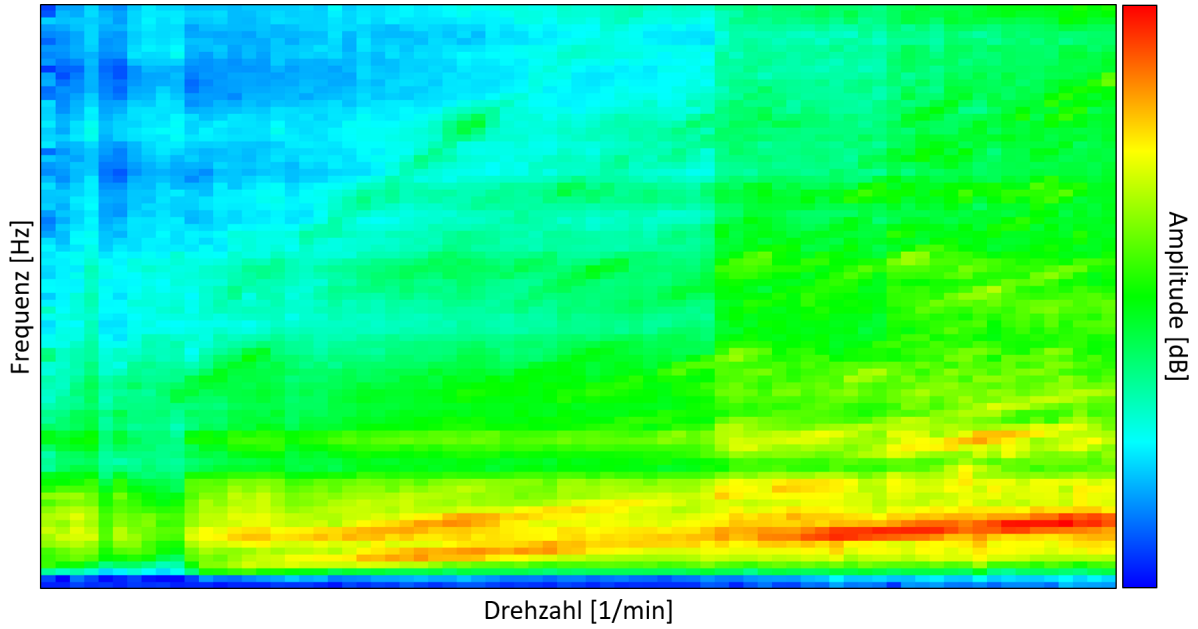


Abbildung 1.2: Beispiel eines Campbell-Spektrogrammes

Aufgrund der aufwendigen Messmethoden und der Auswertung sind solche präzisen Messungen kostenintensiv und räumlich an einen Akustikprüfstand gebunden. Allerdings kann eine grobe Messung bereits Erkenntnisse zu Problemen in den Schwingungseigenschaften des Motors geben. Eine solche grobe Messung ist weniger kostenintensiv, da die Messmethoden einfacher gestaltet sein können, und muss auch nicht an einen Akustikprüfstand gebunden sein. Dieser lässt sich durch ein mobiles Endgerät wie ein Smartphone ersetzen. Es gibt bereits Applikationen für NVH-Messungen auf dem Markt [1], diese sind aber proprietär und haben dementsprechend hohe Lizenzkosten.

Somit ist das Ziel dieser Arbeit, eine frei erhältliche Software für mobile Endgeräte, welche kosteneffizient Schalldaten eines Kraftfahrzeugs sowie dessen Motordrehzahl aufzeichnet und daraus ein Campbell-Spektrogramm, welches Drehzahl, Frequenz und Amplitude grafisch in Verbindung setzt, erstellt, als ein Produkt für einen Endkunden zu entwickeln.

2 Grundlagen

2.1 Mathematische Grundlagen

2.1.1 Signaltransformationen zwischen Zeit- und Bildbereich

Die meisten Signale liegen nach der Messung im sogenannten Zeitbereich vor. Dabei wird die relevante Messgröße, wie z.B. ein Schalldruck oder eine Spannung, als Funktion über den jeweiligen Messzeitraum abgebildet.

Die Darstellung von Signalen im Zeitbereich eignet sich insbesondere zur zeitlichen Einordnung von spontanen, aperiodischen Ereignissen. Dabei können verschiedene charakteristische Änderungen der Messgröße, z.B. Messspitzen oder -flanken, als Indikatoren für das Auftreten spezifischer physikalischer Vorgänge interpretiert werden.

Bei vollständig oder teilweise periodischen Vorgängen wie z.B. Schwingungen kann es jedoch interessant sein, das Signal im sogenannten Frequenz- oder Bildbereich zu betrachten. Dabei werden die Anteile einzelner Frequenzen am Gesamtsignal über das gesamte Frequenzspektrum abgebildet.

Um ein gemessenes Signal vom Zeit- in den Bildbereich zu überführen, muss es transformiert werden.

2.1.1.1 Fourier-Transformation

Sei $y(t)$ eine stetige, integrierbare Funktion, so kann sie mittels der **Fourier-Transformation** [2] in ihre Fourier-Transformierte $Y(f)$ überführt werden, die den Anteil einzelner Frequenzen f_k am ursprünglichen Signal $y(t)$ über das Frequenzspektrum f abbildet.

2.1.1.2 Diskrete Fourier-Transformation

Messgrößen können in der Praxis meist nicht zeitkontinuierlich gemessen werden. Stattdessen werden sie in diskreten, üblicherweise äquidistanten Zeitabständen abgetastet. Das Verhalten des Signals zwischen zwei Messzeitpunkten ist dabei unbekannt.

Das erhaltene Messsignal liegt somit nicht als stetige, integrierbare mathematische Funktion sondern als Folge von Messwerten vor.

Im Bezug auf das ideale kontinuierliche Signal $y(t)$ gilt für das diskrete Signal $y^*(t)$ und das Abtastintervall t_A der Zusammenhang

$$y^*(t) = y(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - n \cdot t_A),$$

dabei gilt $n \in \mathbb{N}$ und

$$\delta(t) = \begin{cases} \infty, & \text{für } t = 0 \\ 0, & \text{für } t \neq 0 \end{cases},$$

$$\int_{-\infty}^{\infty} \delta(t) = 1.$$

$\delta(t)$ wird als **Dirac-Impuls** [2] bezeichnet. Es handelt sich um eine Funktion, die einen unendlich hohen Funktionswert annimmt, sollte ihr Argument 0 sein. Ansonsten ist ihr Funktionswert 0. Die Fläche unter einem Dirac-Impuls ist immer genau 1.

Der Term $\sum_{n=-\infty}^{\infty} \delta(t - n \cdot t_A)$ erzeugt einen Kamm an Dirac-Impulsen.

Damit sagt die Gleichung aus, dass das diskrete Signal $y^*(t)$ genau das Produkt des idealen Signals $y(t)$ und einem Kamm aus Dirac-Impulsen ist. Dies führt dazu, dass das diskrete Signal zu allen Abtastzeitpunkten $n \cdot t_A$ exakt den Wert des kontinuierlichen Signals zu dem jeweiligen Zeitpunkt, ansonsten den Wert 0 hat.

Aufgrund der Tatsache, dass es sich beim Signal nun um eine Folge handelt, kann die in Kap. 2.1.1.1 beschriebene Fourier-Transformation nicht angewendet werden.

Stattdessen kann hier jedoch auf die **diskrete Fourier-Transformation** [2] zurückgegriffen werden.

Es ist zu beachten, dass die Gleichung zwar das zeitkontinuierliche Signal $y(t)$ beinhaltet, jedoch werden dessen Funktionswerte nur an Stellen betrachtet, die vom diskreten Signal erfasst werden. Somit ist das kontinuierliche Signal zur Ermittlung der Frequenzanteile nicht zwingend erforderlich.

2.1.2 Interpolation

Sind von einer physikalischen Kennlinie weniger Messpunkte bekannt als für eine bestimmte Anwendung erforderlich, so kann unter der Annahme, dass die bekannten Messpunkte zuverlässig sind, die **Interpolation** [3] zur Bestimmung der unbekannten Zwischenwerte genutzt werden.

Bei diesem Verfahren werden die bekannten Messpunkte (u_k, y_k) als Stützstellen für eine Modellfunktion verwendet, an der die unbekannten Werte abgelesen werden können.

Sind die gemessenen Werte nicht zuverlässig, d.h. mit mindestens einer Störgröße überlagert, so wird stattdessen auf die mathematische Methode der **Approximation** [3] zurückgegriffen.

2.1.2.1 Polynominterpolation

Eine Möglichkeit der Interpolation ist die Darstellung aller Messwerte durch ein einziges Polynom $\hat{y}(u)$. Für n gegebene Messpunkte erhält man ein Polynom der Form

$$\hat{y}(u) = \sum_{i=0}^{n-1} a_i u^i$$

mit Grad $n - 1$.

Im Sonderfall, dass es genau zwei Messpunkte (u_0, y_0) und (u_1, y_1) gibt, spricht man von **linearer Interpolation** [3]. $\hat{y}(u)$ ist in diesem Fall eine lineare Funktion und kann über die Vorschrift

$$\hat{y}(u) = y_0 + \frac{y_1 - y_0}{u_1 - u_0}(u - u_0)$$

berechnet werden.

Ein Hauptvorteil dieses Ansatzes sind die Stetigkeit, sowie die vollständige Differenzierbar- und Integrierbarkeit von $\hat{y}(u)$.

Der Hauptnachteil dieser Methode ist, dass sie für größere Punktemengen n ungeeignet ist. Funktionen zu hohen Grades können unerwünschtes Schwingverhalten zwischen den einzelnen Stützstellen aufweisen.

2.1.2.2 Spline-Interpolation

Um das in Kap. 2.1.2.1 beschriebene Oszillationsverhalten zu vermeiden, wird bei größeren Punktemengen n häufig die **Spline-Interpolation** [3] eingesetzt.

Bei dieser Methode wird zwischen jedem Punktepaar $(u_n, y_n); (u_{n+1}, y_{n+1})$ ein separates Polynom $\hat{y}_n(u)$, ein sogenanntes **Spline** berechnet.

Splines sind dabei Polynome, die nur zwischen ihren Randpunkten definiert sind und dabei gewisse Randbedingungen erfüllen müssen.

So müssen der Anfangs- und Endwert des Splines exakt auf den Randpunkten liegen. Außerdem muss ein Spline in seinen Randpunkten die selbe Steigung wie sein benachbartes Spline haben. Damit ergibt sich über den gesamten Messbereich eine stetige, mindestens einmal stetig differenzierbare Funktion, mit der Zwischenwerte interpoliert werden können.

Anders als bei der Polynominterpolation sind die einzelnen Splines Funktionen recht niedrigen (typischerweise zweiten oder dritten) Grades. Damit wird einem Oszillieren zwischen den Messpunkten vorgebeugt.

2.2 Technische Grundlagen

2.2.1 Android-Plattform

Android ist ein freies mobiles Betriebssystem, das unter der Mitwirkung von Google entwickelt wird. Für Android können Software-Applikationen (Apps) entwickelt werden, die dank dem Android-Marktanteil von 74% (Stand Juni 2020) [4] einem großen Teil der Mobilnutzer zugänglich sind.

2.2.2 OBD II-Schnittstelle

OBD II (On-Board-Diagnose, Revision 2) ist die Abkürzung für ein Fahrzeugdiagnosesystem, das in der Europäischen Union seit dem Jahr 2004 für Benzin- und Diesel-PKW-Neufahrzeuge vorgeschrieben ist [5]. Entsprechend ausgestattete Automobile besitzen eine **OBD-2-Schnittstelle**, die in der Regel direkt mit dem jeweiligen internen Steuerungssystem verbunden ist. Die Kommunikation erfolgt dabei normalerweise über den sogenannten *CAN-Bus* des Autos, über den viele Komponenten der Fahrzeugelektronik miteinander kommunizieren – darunter auch die Motorsteuerung, die die für eine Akustikanalyse relevante Motordrehzahl zur Verfügung stellen kann.

Abbildung 2.1: OBD II-Schnittstelle im Handschuhfach eines Renault Twingo II

Für die Verbindung mit der OBD-Schnittstelle eines Autos existieren verschiedene (kabelgebundene und kabellose) Adapter (im letzteren Fall sind das sogenannte **Dongles**) – darunter auch solche, die sich über Bluetooth ansteuern lassen (zur Umsetzung siehe Abschnitt 3.2.3.1).

2.2.3 Bluetooth

Bluetooth ist ein Standard zur digitalen und drahtlosen Datenübertragung, der für kurze Übertragungsdistanzen optimiert ist. Fast alle Smartphones und Tablets besitzen heutzutage ein entsprechendes Sende- und Empfängermodul – die Technik wird beispielsweise für Musikstreaming, Indoor-Navigation [6], die Nachverfolgung von Begegnungen zum Zwecke der Eindämmung von SARS-CoV-2 [7], oder für die Verbindung mit den bereits erwähnten OBD-Dongles eingesetzt.

Bevor zwei Bluetooth-Geräte Anwendungsdaten miteinander austauschen können, müssen sie durch den Nutzer aktiv miteinander gekoppelt werden („Pairing“) (Ausnahme: Bluetooth Low Energy). Während dieses Prozesses wird die eindeutige und 48 Bit lange Identifizierungsnummer des jeweils anderen Bluetooth-Geräts, die *MAC-Adresse*, abgespeichert. Im Falle von Android (und iOS) wird dafür eine grafische Benutzeroberfläche vom Betriebssystem zur Verfügung gestellt. Die Kopplung des OBD-Dongles muss daher nicht direkt durch eine App erfolgen, die das Bluetooth-Modul verwendet.

Der Bluetooth-Standard verfolgt eine Master-/Slave-Architektur - das heißt, dass nur ein Teilnehmer (der *Master*) des Bluetooth-Übertragungskanals eine Datenübertragung initiieren kann. Im Falle der Kommunikation zwischen Android-Gerät und OBD-Dongle nimmt diese Rolle das Android-Gerät ein. Alle anderen Teilnehmer werden als *Slave* bezeichnet.

2.2.4 CSV-Format

Das CSV-Format (**C**omma **S**eparated **V**alues) ist ein Datenformat, mit dem tabellarische Daten in strukturierter Form verarbeitet und gespeichert werden können. CSV-Dateien sind Textdateien, in denen jede Textzeile für eine Tabellenzeile steht und die Datenfelder (Spalten) innerhalb einer Zeile durch Kommata (oder andere, festgelegte Zeichen) abgetrennt werden.

Eine gültige Datei im CSV-Format könnte zum Beispiel so aussehen:

```
1 Name,Vorname,Klasse
2 Mueller,Luca,6a
3 Schmidt,Lisa,7b
```

Listing 1: Eine gültige CSV-Datei

2.2.5 Audioaufzeichnung

Der mit einem Mikrofon in elektrische Signale umgewandelte Luftschall kann in digitaler Form aufgezeichnet und in einer **Audiodatei** abgespeichert werden. Es folgt eine kurze Übersicht über die wichtigsten Grundlagen dieses Verfahrens, die für die Umsetzung des Projekts von Bedeutung sind.

2.2.5.1 Audiocodec

Ein **Audiocodec** ist ein Algorithmenpaar bestehend aus **Encoder** und **Decoder**, das Audiosignale codieren und decodieren kann. Audiocodecs werden überall dort benötigt, wo

Audiodaten in digitalen System verarbeitet werden - also natürlich auch auf Smartphones und Tablets.

Beim **Codieren** wird dabei das analoge Signal (z. B. von einem Mikrofon) in einen digitalen Datenstrom umgewandelt, während beim **Decodieren** ein Datenstrom wieder in ein analoges (Lautsprecher-)Signal überführt wird.

Viele Audiocodecs setzen zudem **Kompressionsalgorithmen** ein, um die Datenmenge des codierten Signals zu reduzieren. Zu unterscheiden sind hier **verlustfreie** und **verlustbehaftete** Verfahren.

Wird ein verlustfreies Verfahren eingesetzt, kann die Datenmenge unter Ausnutzung typischer Merkmale eines Audiosignals reduziert werden - die exakte Rekonstruktion des codierten Signals bleibt dabei möglich. Als Steigerung dazu machen sich verlustbehaftete Verfahren Erkenntnisse der **Psychoakustik** zu Nutze, um die Datenmenge noch weiter zu vermindern. Das Ausgangssignal kann dann zwar nicht mehr Bit für Bit rekonstruiert werden, ein Mensch kann aber häufig nur einen geringen bis keinen Unterschied zum Ursprungssignal feststellen.

Es ist möglich, komplett auf die Komprimierung von Audiodaten zu verzichten. Sie liegen dann meist in Form der **Puls-Code-Modulation (PCM)**-Rohdaten vor. Genauso wie beim Einsatz verlustfreier Kompressionsverfahren hängt dann die Qualität der Daten nur noch von der **Abtastrate** und der **Bittiefe** ab - eine entsprechende Qualität des analogen (Mikro- fon-)Signals vorausgesetzt.

2.2.6 Activity

Eine **Activity** ist eine Android-App-Komponente, die dem Nutzer eine einzelne grafisch bedienbare Funktionalität bereitstellt und eine bestimmte Aktion repräsentiert - zum Beispiel die das Anzeige einer Nachricht in einer E-Mail-App oder die Bearbeitung eines Termins in einer Kalender-App.

Das Fenster einer Activity erstreckt sich normalerweise über den gesamten Bildschirm.

Activities werden durch Java-Klassen repräsentiert, die Kindklassen der Klasse *Activity* sind. Die Klasse *Activity* selbst ist Teil des Android-Frameworks.

Jede Activity unterliegt einem Lebenszyklus, für den mehrere Zustände (wird gestartet, wird ausgeführt, wird beendet, ...) definiert sind. Startet, minimiert oder oder schließt der Nutzer (oder das System) die Aktivität, findet ein Zustandswechsel statt. Activities können hierauf reagieren, wenn ihre sogenannten **Callback-Methoden** (Rückrufmethoden) definiert werden. Beispielsweise wird die Callback-Methode *onCreate()* einer Activity sofort aufgerufen, wenn die Activity gestartet wird - hier werden in der Regel Instanzvariablen initialisiert, die Benutzeroberfläche gesetzt und ggf. ein gespeicherter Zustand der Activity wiederhergestellt [8, S. 123, 129 f.][9].

2.2.7 Service

Ein Service („Dienst“) ist eine Android-App-Komponente, mit der Hintergrundaufgaben ausgeführt werden können. Ein **Foreground Service** kann selbst dann ausgeführt werden, wenn der Nutzer nicht mit der App interagiert, die den Service bereitstellt, und muss eine Benachrichtigung anzeigen [10].

Activities können sich an Services binden, um an diese Anfragen zu senden und von ihnen Rückgabewerte zu erhalten [11]. Damit kann beispielsweise eine Aufzeichnung oder eine Musikwiedergabe, die in einem Service ausgeführt wird, durch grafische Bedienelemente einer Activity überwacht und gesteuert werden.

Wenn Android-Dienste explizit gestartet werden, werden sie unabhängig vom Lebenszyklus einer Activity ausgeführt.

2.2.8 Datenstrukturen

2.2.8.1 Array

Ein Array, auch **Feld** genannt, ist eine informatische Datenstruktur, welche eine Mehrzahl gleichartig strukturierter Daten enthält (Beispiele: Ganzzahlen, Zeichenketten). Die Strukturierung der Dateneinträge erfolgt in beliebig vielen Dimensionen. Der Zugriff auf jedes Objekt kann durch Angabe seiner Indizes erfolgen, die seine Position im Array beschreiben. Zweidimensionale Arrays können auch als **Tabelle** aufgefasst werden - die beiden Indizes entsprechen dann der Zeile und der Spalte des adressierten Objekts.

2.2.8.2 Liste

Eine Liste ist eine dynamische Datenstruktur, welche wie ein Array gleichartig strukturierte Daten enthält. Listen sind Arrays allgemein sehr ähnlich: Ihre Strukturierung erfolgt in beliebig vielen Dimensionen und sie weisen jedem Objekt wie in einer Matrix eindeutige Indizes zu. Umgekehrt können die Objekte auch wieder anhand ihrer Indizes aufgerufen werden. Eine Besonderheit der Liste ist, dass die Anzahl der Objekte variieren kann – auch nach der Initialisierung der Liste. Somit ist es möglich, immer mehr Objekte zur Laufzeit des Programms hinzuzufügen oder zu entfernen, wobei sich die Grenzen der Liste dynamisch anpassen.

3 Umsetzung

3.1 Entwicklungsumgebung

Die App wurde mit dem offiziellen Android Software Development Kit (SDK) und der Programmiersprache Java entwickelt, da sich diese Lösung im Vergleich zu App-Baukästen wie wie *MIT AppInventor* deutlich flexibler einsetzen lässt.

GitHub wurde als Kollaborationsplattform gewählt.

3.2 Datenerfassung

3.2.1 Hauptaktivität (*MainActivity*)

Die Akustik-App besteht zum jetzigen Entwicklungsstand aus einer einzelnen Activity (*MainActivity*, siehe Kap. 2.2.6), die zum Starten und Stoppen der Aufzeichnung verwendet werden kann. Das zu verwendende Bluetoth-Gerät kann hier ebenfalls durch Klick auf die entsprechende Schaltfläche ausgewählt werden.

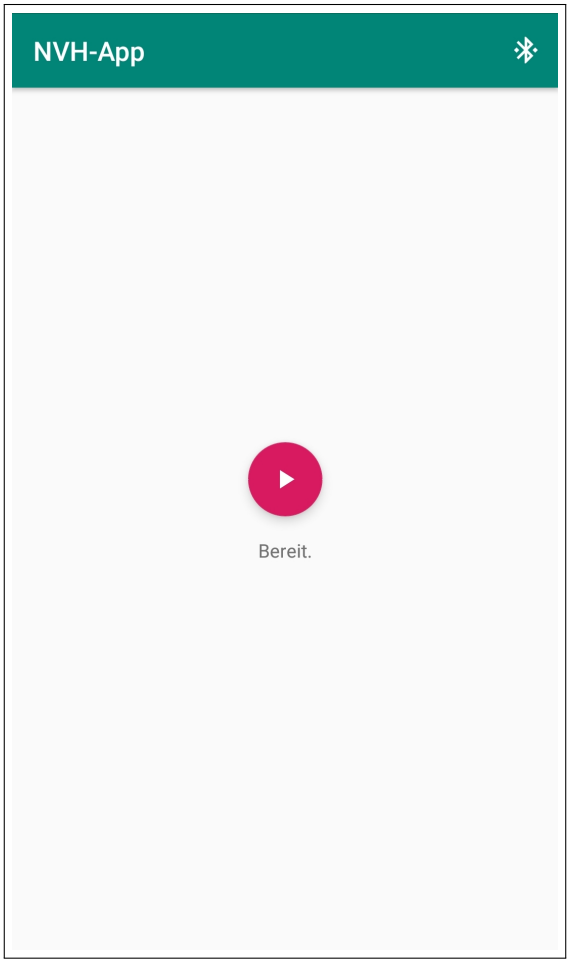


Abbildung 3.1: Der MainActivity-Bildschirm

3.2.2 Vordergrunddienst (*DataRecordingService*)

Da die Audio- und Drehzahldatenaufzeichnung unabhängig vom Lebenszyklus der Hauptaktivität und auch dann möglich sein soll, wenn gerade eine andere App geöffnet ist, finden die entsprechenden Abläufe innerhalb eines **Foreground Service** statt (siehe Kap. 2.2.7) - dem *DataRecordingService*. Enstprechend den Android-Vorgaben wird während seiner Ausführung eine Benachrichtigung angezeigt (siehe Bild).

Die Aufzeichnung der Daten wird durch die *MainActivity* gesteuert, die sich an den Service bindet (siehe auch Kap. 2.2.7).

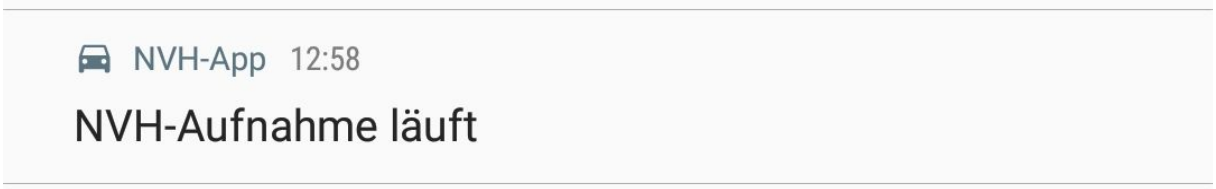


Abbildung 3.2: Die Benachrichtigung des DataRecordingService

3.2.3 Drehzahldaten

3.2.3.1 OBD-Adapter

Ursprünglich sollte die OBD-Schnittstelle des Autos mit einem Arduino-Board abgefragt und die Daten über ein *HM-10 Bluetooth Low Energy (BLE)*-Modul bereitgestellt werden. Dieser Ansatz wurde jedoch bereits in der Planungsphase verworfen, da mit einem frei erhältlichen Bluetooth-OBD-Dongle eine ähnliche Funktionalität mit deutlich reduziertem Entwicklungsaufwand erreicht werden konnte.

Konkret wurde ein *ELM327*-kompatibler Bluetooth-OBD-Adapter verwendet (siehe Bild), der je nach Quelle bereits für unter zehn Euro erhältlich ist.

Abbildung 3.3: Der verwendete ELM327-kompatible OBD-Adapter

Für die Kommunikation mit ELM327-kompatiblen Bluetooth-Adaptern und die Abfrage der Motordrehzahl in Java-Projekten existieren diverse freie Softwarebibliotheken, darunter auch die in diesem Projekt eingesetzte Lösung *obd-java-api* [12]. Obwohl die Entwicklung der Bibliothek bereits seit rund drei Jahren eingestellt ist, erwies sie sich als sehr zuverlässig.

3.2.3.2 Ansteuerung des Adapters im Programmcode

Das Bluetooth-Modul eines Android-Geräts wird über die Java-Klasse *BluetoothAdapter* repräsentiert, die durch das Framework bereitgestellt wird. Speziell für OBD-Dongles wird die Klassenmethode *createInsecureRfcommSocketToServiceRecord(UUID uuid)* benötigt, mit der eine serielle Verbindung zum Dongle über das Bluetooth-Profil Serial Port Profile (SPP) aufgebaut werden kann. [1] Die Methode liefert ein *BluetoothSocket*-Objekt zurück, mit dem die Java-OBD-Bibliothek Befehle an das Dongle senden kann, die dann wiederum an den CAN-Bus des Autos weitergeleitet werden. Rückgabewerte werden auf umgekehrtem Weg zurückgeliefert.

Die ständige Datenerfassung der Drehzahldaten sollte in der App über eine periodische Abfrage des Dongles im Hintergrund („**Polling**“) erfolgen - anschließend sollten die Messpunkte mit einem Zeitstempel versehen und im CSV-Format gepspeichert werden. Polling ist hier notwendig aufgrund der Master-/Slave-Architektur des Bluetooth-Protokolls (siehe Kap. 2.2.3).

Da für diese Aufgaben keine frei verfügbare (und zufriedenstellende) Lösung existierte, wurde der entsprechende Programmteil vollständig neu implementiert. Die Datenerfassung und -speicherung erfolgt dabei in der Thread-Klasse *ObdRecordingThread*, deren *run*-Methode im Hintergrund ausgeführt wird. Innerhalb der Methode werden die Motordrehzahlen periodisch vom OBD-Adapter abgerufen und wie vorgesehen im CSV-Format abgespeichert.

Übertragungsfehler, die durchaus auftreten können, werden hier ebenfalls abgefangen; bei Bedarf wird eine neue Verbindung zum Adapter aufgebaut.

[1] Die UUID (universell eindeutige Identifizierungsnummer), die für die Kommunikation mit einem seriellen SPP-Bluetooth-Gerät wie dem Dongle benötigt wird, lautet *00001101-0000-1000-8000-00805F9B34FB*.

3.2.4 Schalldaten

3.2.4.1 Berechtigungen

Seit der Android-Version 6 müssen Apps die Nutzung bestimmter Funktionalitäten vom Nutzer genehmigen lassen, worunter auch der Zugriff auf das Gerätemikrofon zählt. Die Akustik-App kommt dieser Anforderung nach, sobald die Aufnahme durch den Nutzer gestartet wird.

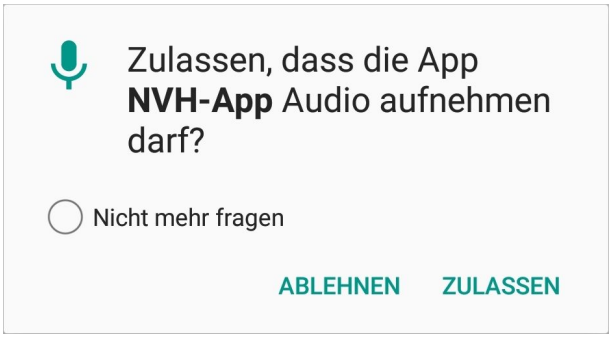


Abbildung 3.4: Der Android-Dialog zur Erteilung der Mikrofonberechtigung

3.2.4.2 Audioaufzeichnung mit dem Android-Framework

Das Android-Framework stellt die Klasse *MediaRecorder* bereit, mit der auf einfache Weise Audiodaten in Form einer Audiodatei aufgezeichnet werden können. Die Kompressionsalgorithmen der unterstützten Codecs (Kap. 2.2.5.1) *AAC*, *AMR*, *Opus* und *Vorbis* sind dabei aber allesamt verlustbehaftet. Für Anwendungsfälle wie ein Diktiergerät kann das in Kauf genommen werden - nicht, wenn die Audiodateien wie in unserem Fall später für eine Akustik-Analyse verwendet werden sollen.

Für diesen und andere Anwendungsfälle existiert die Android-Klasse *AudioRecord*, die den direkten Zugriff auf die Zeit-Amplituden-Daten des Mikrofons ermöglicht, ohne vorher verlustbehaftet zu komprimieren. Im Gegensatz zur MediaRecorder-Klasse erfolgt allerdings keine Aufzeichnung im Hintergrund in eine Audiodatei, sondern es muss periodisch in einem selbst zu erstellenden Hintergrund-Thread die *read*-Methode aufgerufen werden, die jeweils die letzten gemessenen Amplituden des Mikrofons zurückliefert. Auch auf die Einhaltung der Spezifikation des zu wählenden Audiodateiformats muss selbst geachtet werden.

Um keine eigene Lösung mit entsprechendem Entwicklungsaufwand und Bug-Risiko entwickeln zu müssen, kam zur Erfüllung dieser Aufgaben die Bibliothek **OmRecorder** [13] zum Einsatz, die sich auf einer ähnliche hohe Abstraktionsebene wie die MediaRecorder-Klasse befindet und trotzdem die Aufzeichnung im unkomprimierten Audioformat **RIFF WAVE** (Dateiendung .wav) zulässt. Intern verwendet OmRecorder dafür die AudioRecord-Klasse von Android.

WAV-Dateien bieten in unserem Anwendungsfall einen weiteren Vorteil: Sie können direkt von der eingesetzten FFT-Bibliothek *QuiFFT* [14] eingelesen werden, es werden also keine Konvertierungsschritte notwendig (siehe Kap. 3.3.2).

3.2.5 Überblick: App-Komponenten für die Datenaufzeichnung

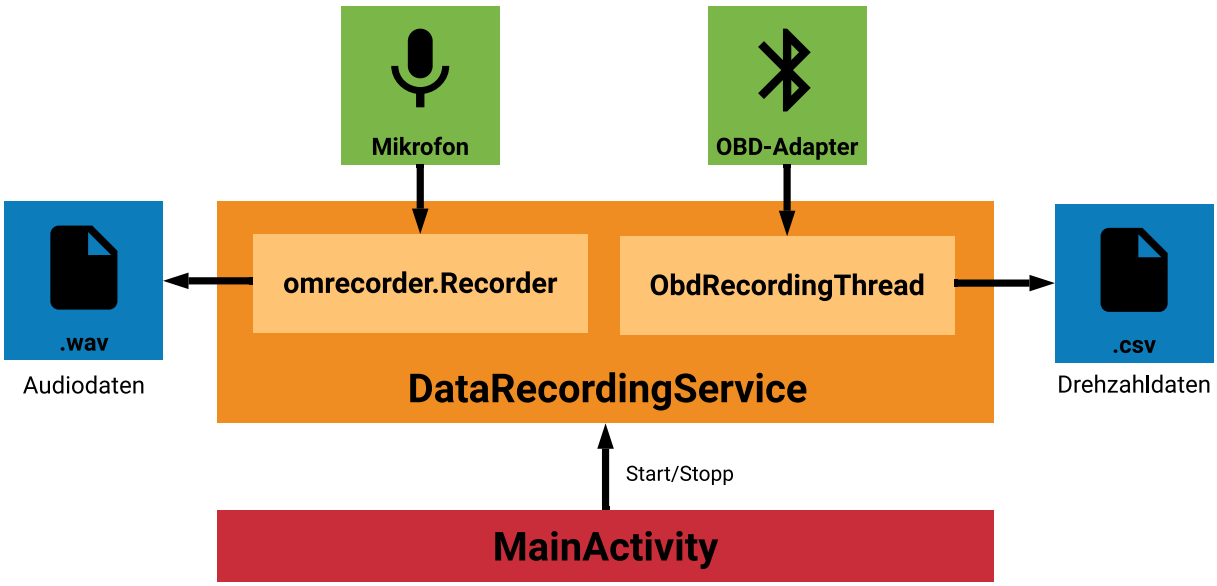


Abbildung 3.5: App-Komponenten für die Datenaufzeichnung

3.2.6 OBD-Simulator-App

Während der Entwicklung von Softwareprojekten ist es oft erforderlich, kleine und große Änderungen zu testen. Insbesondere bei Änderungen am Programmcode, die mit der OBD-Schnittstelle zusammenhängen, waren solche Tests bei uns zunächst sehr aufwändig - denn der OBD-Dongle funktioniert nur an einem echten Auto.

Abhilfe schaffte die kostenlos erhältliche App *OBDII Simulator* [15], mit der auf einem zweiten Android-Gerät ein Auto mit OBD-Dongle simuliert werden kann. Der Quellcode dieser App ist leider nicht öffentlich verfügbar - eine brauchbare Alternative konnte aber nicht ermittelt werden.

3.3 Auswertung

Die aufgezeichneten Schall- und Drehzahldaten werden ausgewertet und in einem Campbell-Spektrogramm visualisiert. Datenerfassung, Auswertung und Visualisierung erfolgen zum jetztigen Entwicklungsstand nacheinander, da dies die naheliegenste Lösung ist.

Das Ziel der Auswertung ist es, die während der Aufzeichnung entstandenen Schall- und Drehzahldaten, welche im WAV- bzw. CSV-Format gespeichert sind, in eine Form zu überführen, bei der jeder Drehzahl die Frequenzen der bei ihr stattfindenden Schwingungen mit ihrer jeweiligen Amplitude zugeordnet werden. Somit ergibt die Auswertung eine dreidimensionale Zuordnung. Dabei ist zu beachten, dass die Eingangsdaten zeitlich abhängig sind. Die Ausgangsdaten dagegen sind zeitlich unabhängig und können direkt an die Klasse, welche für die Visualisierung zuständig ist, übergeben werden.

3.3.1 Analyse der Drehzahldaten

Die CSV-Datei mit den Drehzahldaten wird ausgelesen und in die Liste *rotationInfo* vom Typ *RotationDataPoint* gespeichert. Wie in Kap. 3.2.3.2 beschrieben, wird die Drehzahl des Fahrzeugs periodisch abgefragt. Die resultierende CSV-Datei besitzt zwei Spalten und so viele Zeilen wie Abfragen. Pro Zeile wird jedem Zeitstempel eine Drehzahl zugeordnet. Dementsprechend besitzt jedes Objekt der Klasse *RotationDataPoint* einen Zeitstempel und eine dazugehörige Drehzahl. Die Zeitstempel werden dabei relativ zum ersten gesetzt. Für die Lese- und Schreibvorgänge von CSV-Dateien wird die Bibliothek *opencsv* [16] verwendet.

3.3.2 Analyse der Schalldaten

Die Schalldaten liegen im WAV-Format als zeitdiskretes Signal vor. Jedem Zeitpunkt im Messzeitraum wird eine Amplitude zugeordnet. Das Signal liegt also im Zeitbereich vor. Um den Anteil der einzelnen Frequenzen für die spätere Visualisierung zu erhalten, muss es vom Zeitbereich in den Frequenz- bzw. Bildbereich überführt werden. Dies geschieht mittels der Fourier-Transformation (Kap. 2.1.1.1). Da das Signal nicht zeitkontinuierlich, sondern zeitdiskret, ist, muss die Diskrete Fourier-Transformation (Kap. 2.1.1.2) angewendet werden. Hierfür wird die Bibliothek *QuiFFT* [14] verwendet.

Es wird ein Objekt der Klasse *QuiFFT* instanziiert, welches die Audiodatei, Klassenmethoden zur Analyse und diverse Analyseparameter enthält. Die Klassenmethode *fullFFT()* führt eine diskrete Fourier-Transformation der gesamten Audiodatei durch und hat ein Objekt der Klasse *FFTResult* als Rückgabewert.

Ein wichtiges Feld dieses Objektes ist *fftFrames*, welches ein Array vom Typ *FFTFrame* ist. Ein *FFTFrame* (Fast Fourier Transformation Frame, „Diskretes Fourier-Transformationsfenster“) beschreibt einen Zeitabschnitt bzw. ein Zeitfenster des Audiosignals, welches die durch Transformation erhaltenen Frequenzen und deren Amplituden in dem jeweiligen Zeitabschnitt enthält.

Die Freuquenzen und deren Amplituden in einem *FFTFrame*-Objekt werden in einem *bins* („Behälter“) genannten Array vom Typ *FrequencyBin* zusammengefasst. In einem Objekt der Klasse *FrequencyBin* wird die kumulative Amplitude der in einem bestimmten Frequenzabschnitt stattfindenden Schwingungen gespeichert. Somit bilden in einem Zeitfenster alle Objekte vom Typ *FrequencyBin* aneinandergereiht die Amplituden aller Frequenzabschnitte in diesem Zeitfenster. Alle Zeitfenster bilden aneinandergereiht das gesamte transformierte Signal.

3.3.3 Verknüpfung von Schall- und Drehzahldaten

Die Drehzahldaten werden mit den transformierten Schalldaten in der Liste *framesWithSpeeds* vom Typ *RotationalFFTFrame* vereint. Ein Objekt der Klasse *RotationalFFTFrame* besitzt ein *FFTFrame*-Objekt und die zu diesem Zeitpunkt zugehörige Drehzahl. Jede Drehzahl besitzt einen Zeitstempel, wohingegen jedes *FFTFrame* zwei Zeitstempel, einen für Start und einen für Ende des jeweiligen Zeitabschnitts, besitzt. Daher wird der arithmetische Mittelwert dieser beiden ermittelt, um so einen das Zeitfenster beschreibenden Zeitstempel zu erhalten.

Dieser erhaltene Zeitstempel liegt immer zwischen den Zeitstempeln zweier Drehzahlen. Somit muss die dazwischen liegende Drehzahl interpoliert werden (Kap. 2.1.2). Aufgrund der Aufnahme entstandenen hohen Punktmenge wird die Spline-Interpolation (Kap. 2.1.2.2) verwendet, wobei bei jedem Spline eine Gerade durch das Punktepaar gelegt wird. Die gesuchte Drehzahl wird also durch lineare Interpolation (Kap. 2.1.2.1) ermittelt.

4 Diskussion

4.1 Ergebnisanalyse

Die für die App erforderlichen Softwareinhalte wurden provisorisch entwickelt und getestet. Die App verfügt über eine Hauptaktivität (Abb. 3.1), in welcher der Nutzer mithilfe von Schaltflächen das Gerät mit dem OBD-Adapter verbinden und die Aufnahme starten kann (Abb. 3.5). Schalldaten werden aufgezeichnet und in einer WAV-Datei gespeichert, Drehzahldaten werden in einer CSV-Datei gespeichert.

Auswertung und Visualisierung wurden, um gewisse Störfaktoren bei einer direkten Implementierung in der App zu vermeiden, zunächst als Konsolenanwendung für Personal Computer (PC) geschrieben und getestet. Diese wurde mithilfe von App-generierten Dateien getestet und ergab zufriedenstellende Campell-Spektrogramme (siehe Abb. 4.1).

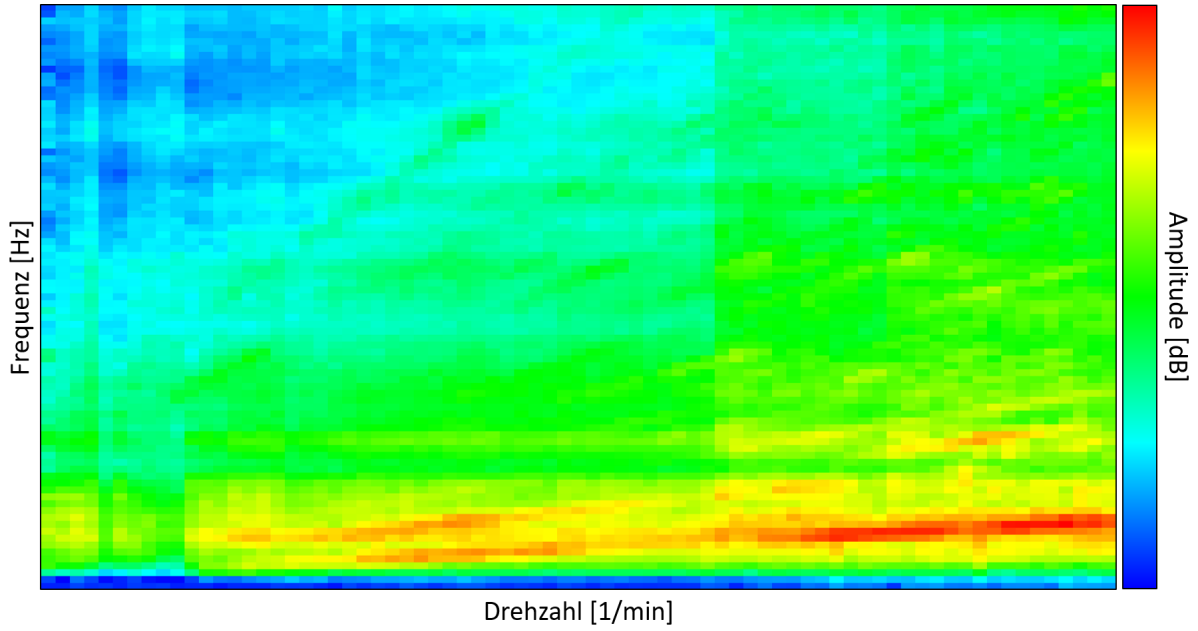


Abbildung 4.1: Campbell-Spektrogramm (mit Konsolenanwendung erstellt)

Der Code für die Auswertung ist in seinen Schnittstellen zu anderen Softwareinhalten so ausgearbeitet worden, dass er direkt in der App implementiert werden kann. Im Bereich der Visualisierung kann der Code für die Berechnung der Farbgradienten übernommen werden. Der tatsächliche Zeichnungsvorgang muss in der App neu implementiert werden.

4.2 Fazit und Ausblick

Das App-Projekt befindet sich zum Zeitpunkt der Abgabe dieser Dokumentation in einem Proof-of-Concept-Stadium, mit dem die grundsätzliche Machbarkeit des Vorhabens gesichert ist und Grundlagen erarbeitet sind. Es wurde provisorischer Quellcode verfasst, der alle Kernfunktionalitäten der App abdeckt.

Die weiteren Realisierungsschritte bestehen im Wesentlichen darin, die Aufzeichnung und die Analyse in einer App zu vereinen und ein benutzerfreundliches Bedienkonzept zu

entwickeln. Außerdem könnten Möglichkeiten zum Datenexport geschaffen werden, sodass sich die App besser in existierende Entwicklungsabläufe integrieren lässt.

Darüber hinaus sollte eine Evaluation der Messqualität erfolgen - am besten durch einen Vergleich mit Messergebissen, die von einem professionellen Akustikprüfstand stammen. Sollte sich dabei herausstellen, dass die Qualität der Analyse maßgeblich darunter leidet, dass die Drehzahlabfrage über den Bluetooth-Adapter zu langsam oder unzuverlässig ist, kann über kabelgebundene Lösungen nachgedacht werden, die evtl. weniger verzögerte Daten oder mehr Messpunkte liefern. Ferner könnte bei entsprechender Notwendigkeit einer besseren Audiodatenqualität die Nutzung eines externen Mikrofons evaluiert werden.

A Danksagung

Wir bedanken uns herzlich bei Manuel Bopp und Sebastian Lutz vom Institut für Produktentwicklung am Karlsruher Institut für Technologie (IPEK), die uns nicht nur zu diesem Projekt inspiriert, sondern auch eine zügige Einarbeitung in die Grundlagen der Fahrzeugakustik ermöglicht haben und für Fragen zur Verfügung standen.

Ein besonderer Dank geht an unsere Kursleiter, Paul Bischof, Thomas Hermann und Anke Richert, die uns über viele spannende Jahre im Hector-Seminar begleitet und tatkräftig unterstützt haben – weit über fachliche Inhalte hinaus.

Abschließend danken wir Josefine und Dr. Hans-Werner Hector und ihrer Hector-Stiftung für die großzügige Förderung des Hector-Seminars und die Ermöglichung der vielen interessanten Angebote, die wir dort wahrnehmen durften.

Literatur

[1] Vibrate Software Inc. *NVH For Android*. (zuletzt abgerufen am 24.07.2020). Jan. 2019. URL: https://play.google.com/store/apps/details?id=vibratesoftware.jacobkelly.nvh.

[2] Fernando León. *Signale und Systeme*. Berlin: De Gruyter, 2019. ISBN: 978-3110626315.

[3] Fernando León. *Messtechnik : Systemtheorie für Ingenieure und Informatiker*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. ISBN: 978-3-662-44820-5.

[4] StatCounter. *Mobile Operating System Market Share Worldwide*. (zuletzt abgerufen am 24.07.2020). Juni 2020. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[5] Wladimir Gurskij. *OBD-2 Allgemeines, technische Informationen*. (zuletzt abgerufen am 24.07.2020). URL: https://www.obd-2.de/obd-2-allgemeine-infos.html.

[6] Jason; Bluetooth SIG Marcel. *Bluetooth is Getting Precise with Positioning Systems*. (zuletzt abgerufen am 24.07.2020). Feb. 2019. URL: https://www.bluetooth.com/blog/bluetooth-positioning-systems/.

[7] *Exposure Notification - Bluetooth Specification v1.2*. (zuletzt abgerufen am 24.07.2020). Apr. 2020. URL: https://blog.google/documents/70/Exposure_Notification_-_Bluetooth_Specification_v1.2.2.pdf.

[8] Thomas Künneth. *Android 8 - Das Praxisbuch für Java-Entwickler. Inkl. Einstieg in Android Studio*. Bonn: Rheinwerk Verlag GmbH, 2018. ISBN: 978-3-836-26058-9.

[9] *Android-Dokumentation: Understand the Activity Lifecycle*. (zuletzt abgerufen am 25.07.2020). Apr. 2020. URL: https://developer.android.com/guide/components/activities/activity-lifecycle.

[10] *Android-Dokumentation: Services overview*. (zueletzt abgerufen am 24.07.2020). Juni 2020. URL: https://developer.android.com/guide/components/services.

[11] *Android-Dokumentation: Bound services overview*. (zuletzt abgerufen am 26.07.2020). Dez. 2019. URL: https://developer.android.com/guide/components/bound-services.

[12] Pires et al. *obd-java-api*. (zuletzt abgerufen am 24.07.2020). Juli 2017. URL: https://github.com/pires/obd-java-api.

[13] Kailash Dabhi. *OmRecorder*. (zuletzt abgerufen am 25.07.2020). URL: https://github.com/kailash09dabhi/OmRecorder.

[14] Miles Henrichs et al. *QuiFFT*. (zuletzt abgerufen am 24.07.2020). Apr. 2019. URL: https://github.com/mileshenrichs/QuiFFT.

[15] FunRary. *OBDII Simulator*. (zuletzt abgerufen am 26.07.2020). URL: https://play.google.com/store/apps/details?id=com.obdii.simulator.

[16] Glen Smith et al. *Opencsv Users Guide*. (zuletzt abegrufen am 24.07.2020). Mai 2020. URL: http://opencsv.sourceforge.net.

Selbstständigkeitserklärung

Hiermit versichern wir, die vorliegende Arbeit unter der Beratung durch Paul Bischof und Thomas Hermann selbstständig verfasst, alle benutzten Hilfsmittel vollständig angegeben und alles kenntlich zu haben, was aus Arbeiten anderer unverändert oder mit Abänderung entnommen wurde.

Aiman Malek	Ort, Datum
Yannik Tausch	Ort, Datum