STRENG VERTRAULICH

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Dokumentation Bachelorprojekt

Heiko Nöldeke, Marc Bolsch, Pascal Roschkowski, Philipp Otto

Entwurf und Ausarbeitung eines Traffic-Noise-Detector-Prototypen

Fakultät Technik und Informatik
Departement Mechatronik

Faculty of Engineering and Computer Science
Departement of Mechatronic Engineering

Heiko Nöldeke, Marc Bolsch, Pascal Roschkowski, Philipp Otto

Traffic-Noise-Detector-Prototyp

Bachelorprojekt eingereicht im Rahmen der Bachelorprüfung

im Studiengang Mechatronik
am Departement Fahrzeugtechnik und Flugzeugbau
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer: Prof. Dr. Rasmus Rettig

Abgabedatum: 15. Juni 2021

Inhaltsverzeichnis

1 Vorwort

Im Rahmen eines Bachelors Projekts im Fachbereich Mechatronik an der HAW Hamburg, ist eine kleine Gruppe Studierender dazu beauftragt, eine praxisnahe Aufgabe zu lösen. Dazu gehört der Prozess der Produktentwicklung, aber auch die Erschaffung eines Prototypen. Diese Dokumentation dient dazu, die Arbeitsschritte und eine Produktbeschreibung festzuhalten. Auf der Basis der Arbeit von Maximilian Welz führen die vier genannten Autoren unter Leitung von Herrn Prof. Dr. Rasmus Rettig eine Anpassung des <Bezeichnung> an die neue Aufgabenstellung an.

2 Zusammenfassung der Aufgabenstellung

Project Charter: Traffic-Noise-Detector (Lärmblitzer), Bachelor Project - Hr. Otto, Hr. Nöldeke, Hr. Bolsch, Hr. Roschkowski

Mission: Aufbau eines Prototypen zur Erkennung und Lokalisierung von Fahrzeigen mit (zu) hoher Lärm Emission (z. B. https://www.auto-motor-und-sport.de/verkehr/laerm-blitzer-in-europa-fallen-gegen-auto-poser/).

Deliverables (incl. timing):

- Anforderungsentwicklung (Mechanisch, Akustisch, Elektrisch, Algorithmisch)
- Systematische Auswahl / ggf. Kombination oder Weiterentwicklung
- Realisierung
- Test und Bewertung der Eignung
- Überarbeitung basierend auf den Testergebnissen [T0+6 Wochen]
- Abschlussintegration, Demonstration/Vortag und Dokumentation [T0+12 Wochen]

Expected Scope / Approach / Activities:

- Einarbeitung in das Messsystem und die Programmierumgebung
- Einarbeitung in den Stand der Technik von Algorithmen zur Erkennung und Lokalisierung akustischer Signale
- Zielgerichtete Auswahl, Weiterentwicklung / Kombination im Hinblick auf genutzte Hardware sowie die Erkennung mit einer hohen Erkennungsrate

Strategic alignment factors:
Integration in dei Arbeitsgruppe Urban Mobility Lab mit den laufenden Arbeiten

Timeframe/Duration:

- Start 1.10.2020 (Vorbereitung)
- Abschluss 30.3.2021 (gerne früher)

Team Resources:
Nutzung Labor Stiftstraße 69 Raum 109 mit der dort verfügbaren Infrastruktur (Elektronikentwicklung, Software Entwicklung, Server, Löteinrichtung; Kamera, Testfahrzeug, Rechner)

Team Process:

- Reglmäßige Reviews (14 tägig)
- Optional: Teilnahme am Teammeeting

3 Umgebungssituation

In verschiedenen Städten um den Globus gibt es das Problem, dass Automobile teilweise so manipuliert werden, dass sie die maximal zulässige Lautstärke überschreiten. Bisher müssen diese Lautstärkesünder manuell von Polizeistreifen ausfindig gemacht und kontrolliert werden. Ähnlich dem Konzept der Geschwindigkeitskontrolle, mit Messung und Beweisaufnahme durch ein Foto, soll mit dieser Entwicklung eines ersten Prototypen eine automatisierte Erkennung der Lautstärke erfolgen und für den Bußgeldbescheid aufbereitet werden.
Wichtige Anforderungen dafür sind neben einer benutzerfreundlichen Bedienung ist auch eine Konstruktion, die den Einsatz im Straßenverkehr ermöglicht. Dazu kommt vor allem eine gute Transportfähigkeit und Wasserbeständigkeit.

4 Konstruktion des Aufbaus

Für ein ressourcenschonendes Arbeiten ist es von großer Bedeutung neben den Methoden der Produktentwicklung auch diejenigen Materialien zu verwenden, die ohnehin schon vorrätig sind. In den entsprechenden Abschnitten wird noch näher drauf eingegangen. Zu bemerken sei aber noch, dass eine Verwendung der verfügbaren Materialien nicht immer die beste Option ist.

4.1 Basisstativ

In dem Labor „Urban Mobility Lab“ liegt bereits das Fotostativ Manfrotto 055 bereit und kann verwendet werden. Herausfordernd kommt aber hinzu, dass sich ein 1/4“ Gewinde auf der Montageplatte befindet. Die folgenden Möglichkeiten zum Umgang mit dem Gewinde wurden diskutiert:

- Nutenstein mit 1/4“ Innengewinde
- Adapterplatte zwischen Montageplatte und Trägerprofil
- Tausch der 1/4“ Schraube durch eine Schraube mit M4 Außengewinde
- Tausch des Stativs

Nach einer Anfrage bei item Industrietechnik GmbH nach Nutensteinen mit entsprechendem 1/4“ Innengewinde, war bekannt, dass sich derartige Produkte auf dem Markt nicht ohne Weiteres besorgen lassen.
Der Tausch des Stativs durch ein alternatives, ebenfalls zur Verfügung stehende, wurde ausgeschlossen, da die Dimensionen des Stativs eindeutig zu massiv sind und der Anforderung einer leichten Transportierbarkeit im Wege stehen.
Eine Adapterplatte aus einem Aluminium-Flachprofil wurde ausgeschlossen, da nach einer Untersuchung des Fotostativs sich herauskristallisiert hat, dass die eingebaute UNC Schraube einfach durch eine M4 Schraube getauscht werden kann.

4.2 Mikrofongehäuse

Dadurch, dass die zum Einsatz kommenden Mikrofone fertig auf einer Platine (Abbildung 1) verbaut sind, aber damit nicht gegen Regenwasser geschützt sind, mussten im Entwicklungsprozess Gehäuse für die Mikrofone konstruiert und 3D-gedruckt werden. Nach sehr kurzer Diskussion war beschlossen, dass mittels CAD ein Gehäuse konstruiert und im vorhandenen 3D-Drucker gefertigt werden sollen (Abbildung 2).

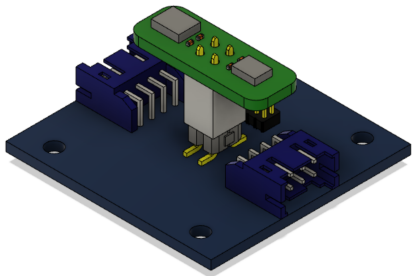

Abbildung 1: Mikrofonplatine

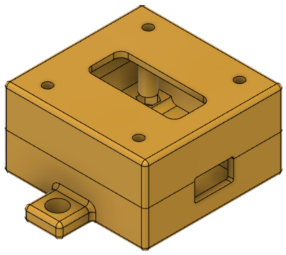

Abbildung 2: Mikrofongehäuse

In der ersten Version dieser Gehäuse (Abbildung 3) soll die Mikrofonplatine über Stifte an Position gehalten und der Gehäusedeckel im Anschluss durch einen Längspressverband mit dem Gehäuseboden verbunden werden.

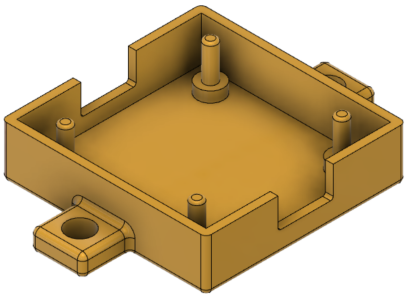

Abbildung 3: Mikrofongehäuse V 0.1

Nach dem Druck der sechs Gehäuse stellte sich heraus, dass:

1. die gedruckten Stifte nicht maßhaltig sind
2. die Löcher im Deckel nicht maßhaltig sind
3. viele druckbedingte Fäden an den Teilen beseitigt werden müssen

An der gedruckten Geometrie lässt sich im Nachgang nicht mehr viel bearbeiten, lediglich die Hülsen im Gehäusedeckel können aufgebohrt werden. Die Erfahrung hat aber gezeigt, dass das verwendete Filament vor allem an den Kontaktflächen der einzelnen Layer zu porös ist und beim Aufbohren sehr schnell bricht. Alle Gehäuse wurden entsorgt.

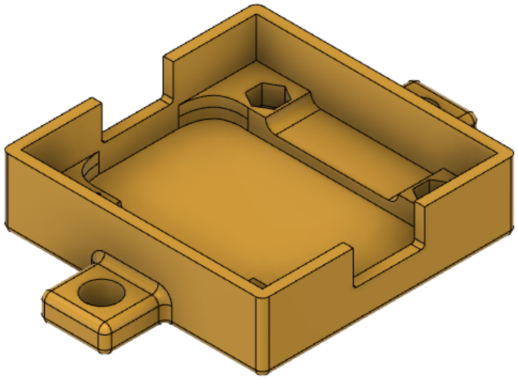

Abbildung 4: Mikrofongehäuse V 0.2

Für die zweite Version (Abbildung 4) der Gehäuse wurden die Stifte im Unterteil und die Hülsen im Deckel entfernt. Im Unterteil sind nun Sechskantaufnahmen für vier M2.5 Muttern und im Deckel Durchgangslöcher für entsprechend Lange Zylinderkopfschrauben. Durch Einpressen und Verkleben werden die Muttern in Position gehalten.
Die Mikrofonplatine wird auf die Flächen auf die Flächen oberhalb der Mutter aufgelegt und das Gehäuse durch Anziehen der vier Schrauben verschlossen. Nach dem Druck eines neuen Prototyps wurde die Brauchbarkeit analysiert und das Gehäuse für die Massenproduktion freigegeben. Nachteilig ist, dass mit ausreichend großer Zugkraft die Gehäusedeckel trotz Verschraubung geöffnet werden können, zudem wirken die vier Schrauben etwas überdimensioniert. In einer zweiten Überarbeitung sollten diese beiden Punkte beachtet werden.

4.3 Unterbringung der Steuereinheit

Für den Raspberry Pi 4, die Raspicam <Modell einfügen> und die Powerbank Powerbank PLUS MacBook 20.100 mAh Space Grey sollte eine vorhandene wasserfeste Installationsbox verwendet werden <Modellbeschreibung einfügen>. Das Hauptproblem dabei war, die zu Powerbank in der Box zu plazieren, daher wurde sich entschlossen, eine neue Installationsbox zu bestellen (Hammond Manufacturing 1555VAL2GY). Für die sichere Positionierung aller Bauteile wird wieder auf die 3D-Druck-Technologie zurückgegriffen und ein Innenleben konstruiert.
Nach der ersten Konstruktion stellte sich heraus, dass die neu besorgte Installationsbox zu niedrig für das Innenleben ist und die USB Kontakte des Rapsberry Pi 4 durch den Deckel stoßen (Abbildung 5). In einem Teamreview wurde eine mögliche Lösung entwickelt, bei der der Raspberry Pi 4 sowohl 180° um die Längs-, als auch 90° im Uhrzeigersinn um die Querachse gedreht, eingebaut werden sollte. Nach der Neukostruktion und der Ergänzung einer Stützkonstruktion für den Raspberry Pi 4, war es möglich in der Installationsbox alle Bauteile zu platzieren, ohne eine erneute Neuanschaffung zu tätigen. (Abbildung 6)

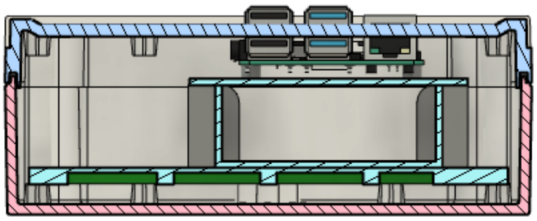

Abbildung 5: Schnitt Installationsbox V 0.1

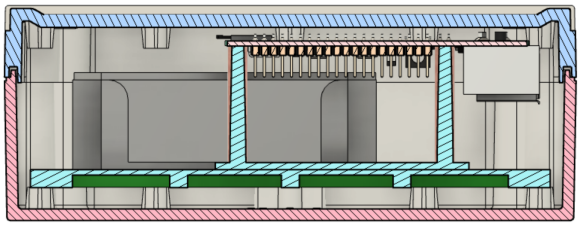

Abbildung 6: Schnitt Installationsbox V 0.2

4.4 Positionierung der Baugruppen entlang eines Arrays

Um die Mikrofone entlang einer Linie positionieren zu können, wurde ein 1,1 m langes vorhandenes item-Profil 6 30x30 eingesetzt, auf dem die Gehäuse der Mikrofone mit Nutensteinen in definiertem Abstand montiert werden können. Dazu wurden am Mikrofongehäuse jeweils rechts und links zwei Befestigungspunkte für M4 Schrauben designt. Als Gegenstück werden wieder Nutensteine eingesetzt.
Das Profil an sich wird, wie oben beschrieben, auch mittels eines Nutensteins auf dem Stativ befestigt. Die Installationsbox wird ebenfalls mit Nutensteinen mittig am Profil befestigt.

5 Programmierung

5.0.1 Beweisführung

Da ist sich bei dem von uns entwickeltem System um einen Prototypen für ein nutzbaren TND handelt, hat eine aussagekräftige Beweisführung der Geräusch-Lokalisierungen eine hohe Priorität. Zum Vorbild genommen wurden dafür herkömmliche Blitzeranlagen aus dem Straßenverkehr, bei denen ein Beweisfoto aufgenommen wird und Metadaten über die gemessene Geschwindigkeit abgespeichert und übermittelt werden.
Das eigentliche Foto wird mithilfe eines „Raspberry Pi 4“ Kameramodul aufgenommen, bearbeitet und mit einem Zeitstempel versehen abgespeichert.
Der hierzu verwendete Programmablauf lässt sich folgendermaßen beschreiben:
Die Beweisführung startet, wenn ein definierter Pegel überschritten wird und die Quelle des Geräusches ermittelt wurde. Da die Kamera nur einen Betrachtungswinkel von 60° aufweist, wird nun überprüft, ob sich die Quelle innerhalb des Betrachtungswinkels, also innerhalb des Kamerabildes befindet[1].
Ist dem der Fall, so wird ein Kamerabild aufgezeichnet und ein Zeitstempel erstellt. Um die Komplexität des Systems möglichst gering halten zu können wird die Kameraansteuerung direkt in dem Hauptprogramm mithilfe einer Wrapper-Klasse realisiert.
Diese nutzt die „RaspiCam: C++ API“ von Rafael Muñoz Salinas, um die Hardware direkt in dem Programmablauf ansteuern zu können[2]. Dies erspart die zusätzliche Verwendung von Unterprogrammen. Das nun aufgezeichnete Bild wird zunächst als „.ppm“ abgespeichert.
In dem nächsten Schritt wird das abgespeicherte Bild wieder geöffnet und mit der Bildverarbeitungsbibliothek „OpenCv“ bearbeitet[3]. Dieser Schritt wird durchgeführt, um die errechneten Quellkoordinaten des Geräusches grafisch darzustellen. Hierbei wird das Bild in eine vorher definierte Anzahl von Segmenten (maximal 60, hier: 7) unterteilt. Die rechnerisch ermittelte Gradzahl der Quelle des Geräusches wird nun auf ein Segment im Bild bezogen. Dieses wird entsprechend farblich mit einem grünen, gut sichtbaren Rechteck markiert.
Im Anschluss wird das bearbeite Bild in einem für Windows-Betriebssysteme einfach zu lesendes Format (.png) abgespeichert. Der zu Beginn der Beweisführung aufgezeichnete Zeitstempel wird nun als Dateiname verwendet.
Die Beweisführung enthält somit: ein Foto des Fahrzeugs, ein markiertes Segment, in dem sich die Quelle des Geräusches befindet und ein Zeitstempel zu welchem Datum und Uhrzeit das Fahrzeug aufgezeichnet wurde. Wahlweise können darüber hinaus auch sämtliche, sich im Buffer befindlichen Daten, die zur Lokalisierung des Geräusches verwendet wurden in Tabellenform abgespeichert werden. Da es sich hierbei aber bei einem längeren Betrieb um erhebliche Datenmengen handelt, die den internen Speicher des „RaspberryWiFi“ belegen würden, wurde darauf bisher verzichtet.

5.1 Programmablauf

Der Programmablauf wird in diesem Abschnitt Schritt für Schritt erklärt und anhand geeigneter Diagramme dargestellt. Zu einer genauen Übersicht des Ablaufes ist der Flowchart des Programmes (**??**) hilfreich. Nun folgt eine chronologische Erklärung der Programmteile.

5.1.1 doaV2

Nach dem Programmaufruf wird die DOA Konfiguration, bestehend aus vorberechneten Werten, ausgelesen. Dies wird durch die doaV2 Klasse mithilfe der loadPrecalculatedValues Methode ermöglicht (**??**).

[1] https://www.raspberrypi.org/documentation/hardware/camera/ zuletzt besucht: 05. 06. 21
[2] https://www.uco.es/investiga/grupos/ava/node/40 zuletzt besucht: 05. 06. 21
[3] https://opencv.org/about/ zuletzt besucht: 05. 06. 21

Hier wird zunächst die Konfigurationsdatei geöffnet. Nun werden für alle möglichen Mikrofonpaare alle vorberechneten Laufzeiten aller möglichen Winkel eingelesen und in das Array precalculatedCompareValues abgespeichert. Zuletzt wird der Zugriff auf die Datei gestoppt. Die DOA Klasse beinhaltet auch die Berechnung des Ursprungsortes eines Geräusches, auf die im späteren Programmverlauf weiter eingegangen wird.

5.1.2 create_pipe

Nach dem Laden der Laufzeiten wird mithilfe der create_pipe Methode ein FIFO angelegt, mit dem die Echtzeitmessungen zur Datenverarbeitung weitergegeben werden.

5.1.3 updateValues

Diese Methode stellt sicher, dass der Daten-Buffer nach jedem FIFO Zyklus neu gefüllt ist. Die Methode bestimmt zuerst das Maximum an gemessenen Werten aus einem Zyklus und schreibt sie dann mithilfe der *fastAddValue* Methode im Signed UInt32 Format in den Speicher

5.1.4 processFFT

Eine einfache „Fast Fourier Transformation“, die ein Array an komplexen Gleitkommazahlen ausgibt. Mithilfe dieser Fouriertransformation ist es möglich, die Intensität der Geräusche von der zwischengespeicherten Aufnahme herauszufinden, um diese zu vergleichen.

5.1.5 getDataForSoundDetect

Im Programmablauf folgt diese einfache Getter-Methode, um die gepufferten Daten aus dem FIFO zur Weiterverarbeitung in ein Array zu schieben.

5.1.6 checkForBang

Die checkForBang Methode ist eine der wichtigsten im gesamten Programmablauf und sitzt direkt in einer Bedingung. Wenn durch diese Methode ein lautes Signal bestätigt wird, fährt das Programm in der Verarbeitung dieses Signales fort.
Wie in **??** zu erkennen ist, wird anfangs mithilfe der calcEnergy Methode aus derselben Klasse der Unterschied zwischen den Energieniveaus von Geräuschen ausgerechnet. Dies geschieht durch die schon erwähnte FFT. Wenn nun die errechnete Differenz den Wert 600000 überschreitet, kann davon ausgegangen werden, dass ein lautes und beobachtungswertes Event stattgefunden hat, da eine sehr hohe Energiedifferenz zu vorherigen Werten gemessen wurde.

5.1.7 getAngle

Diese Methode ist nun dafür verantwortlich, dem Geräuschevent eine eindeutige Richtung zuzuordnen, um eine Beweisführung zu ermöglichen. Sie ruft zwei weitere wichtige Methoden auf – xcorr und compareValues.
Mithilfe der vorberechneten Korrelationswerte für das Mikrofonarray kann die xcorr Methode verwendet werden, um die Korrelation zweier Signale zu finden.
Mit compareValues werden die Korrelationen verglichen und mit einer einfachen Sortierung die stärksten Übereinstimmungen gefunden, um daraus eine Richtung zu erschließen.
Mehr dazu im Abschnitt des Protokolls, dass auf die Berechnung des Winkels näher eingeht.

5.1.8 cameraControl

Die Klasse cameraControl benutzt nun, wenn sich der errechnete Winkel im richtigen Sichtfeld der Kamera befindet, zwei Methoden, um ein Foto zu erstellen und dies so zu bearbeiten, dass der Bereich der Schallquelle sichtbar wird. Mithilfe der einfachen Schnittstelle der RaspiCam

und dazugehörigen Bibliotheken wird ein Bild im RGB-Format von der Kamera angefordert und abgespeichert. Nun wird mithilfe des errechneten Ergebnisses aus getAngle und OpenCV das Bild bearbeitet, indem durch mehrere else-if-Abfragen zum richtigen Bildsegment iteriert wird und dann dieses Segment mit einem Rechteck hervorgehoben wird.

5.2 Ortung der Geräuschquelle

Das Orten einer Schallquelle ist im Gegensatz zum Erfassen eines lauten Geräusches deutlich umfangreicher. Hierzu wird die Kreuzkorrelation zwischen allen Mikrofonen untereinander errechnet um diese als Hilfe zu benutzen, um das mathematische Maximum zu berechnen.
Die Kreuzkorrelation ist eine Messung, die die Bewegungen von zwei oder mehr Sätzen von Zeitreihen-Daten relativ zueinander verfolgt. Sie wird verwendet, um mehrere Zeitreihen zu vergleichen und objektiv zu bestimmen, wie gut sie miteinander übereinstimmen und insbesondere, an welchem Punkt die beste Übereinstimmung auftritt. Die Kreuzkorrelation kann auch jegliche Regelmäßigkeiten in den Daten aufdecken.
Betrachten wir als Beispiel zwei reellwertige Funktionen f und g, die sich nur durch eine unbekannte Verschiebung entlang der x-Achse unterscheiden. Man kann die Kreuzkorrelation verwenden, um herauszufinden, um wie viel g entlang der x-Achse verschoben werden muss, damit es mit f identisch wird. Die Formel verschiebt im Wesentlichen die Funktion g entlang der x-Achse, wobei das Integral ihres Produkts an jeder Position berechnet wird. Wenn die Funktionen übereinstimmen, wird der Wert von $(f \star g)$ maximiert. Das liegt daran, dass Spitzen (positive Bereiche), wenn sie übereinstimmen, einen großen Beitrag zum Integral leisten. Ähnlich verhält es sich, wenn sich Talsohlen (negative Bereiche) ausrichten, leisten sie ebenfalls einen positiven Beitrag zum Integral, da das Produkt zweier negativer Zahlen positiv ist. Bei komplexwertigen Funktionen f und g stellt die Konjugierte von f sicher, dass ausgerichtete Spitzen (oder ausgerichtete Talsohlen) mit imaginären Komponenten positiv zum Integral beitragen.
Kreuzkorrelationen sind nützlich für die Bestimmung der Zeitverzögerung zwischen zwei Signalen, z. B. für die Bestimmung von Zeitverzögerungen für die Ausbreitung von akustischen Signalen über ein Mikrofonarray. Nach der Berechnung der Kreuzkorrelation zwischen den beiden Signalen zeigt das Maximum (oder Minimum, wenn die Signale negativ korreliert sind) der Kreuzkorrelationsfunktion den Zeitpunkt an, an dem die Signale am besten ausgerichtet sind; d. h., die Zeitverzögerung zwischen den beiden Signalen wird durch das Argument des Maximums die Kreuzkorrelation bestimmt. Diese mathematische Eigenschaft benutzen wir um mit $\tau = arg\ _{t\in\mathbb{R}}\ max\left((f \star g)(t)\right)$ den zeitlichen Versatz zu errechnen.
In unserem Fall werden diese Maxima initial in Abhängigkeit der geometrischen Form des Mikrofonarray ausgerechnet. Dann wird in der *xcorr* Methode die maximale Latenz zwischen jeder möglichen Mikrofonpaarung ausgerechnet und diese dann in der compareValues Methode miteinander verglichen. Diese Methode vergleicht nun alle gemessenen Verzögerungen mit den vorher ausgerechneten Verzögerungen, und zwar für jeden möglichen Winkel. Der Winkelbereich mit den größten Übereinstimmungen über alle Mikrofonpaarung, kann mit Sicherheit als der Bereich gekennzeichnet werden, aus dem das Geräusch kommt.

Abschnitt	**Inhalt**
Name	Mikrofonfunktionstest
Autor(en)	Philipp Otto
Priorität	hoch
Kritikalität	hoch
Verantwortlicher	Philipp Otto
Kurzbeschreibung	Mit diesem Test soll die fehlerfreie Funktion der Mikrofone getestet werden
Auslösendes Ereignis	Eingabe Konsole: „sudo ./RecordMicToRAM“
Akteure	Mirkofonpaar, Rasberry Pi, Jumperkabel
Vorbedingung	Raspberry eingerichtet, Mikrofone und Raspberry mit Kabel verbunden, Raspberry eingeschaltet
Nachbedingung	„RecordMicToRAM“ beendet, Prüfung der .txt-Datei ist erfolgt
Ergebnis	.txt- Datei erstellt
Hauptszenario	1. Über die Konsoleneingabe wird das Programm gestartet 2. Das Programm zeichnet ca. 8 Sekunden lang den Aikrofonausgang auf 3. Das Programm erstellt eine .txt-Datei mit allen Messwerten 4. Die .txt-Datei wird auf Vollständigkeit überprüft
Alternativszenario	4.b Die .txt-Datei enthält keine Messwerte 4.c Kabelverbindung wird überprüft

Tabelle 1: Mikrofonfunktionstest

Abschnitt	**Inhalt**
Name	Kabelfunktionstest Nr. 1
Autor(en)	Philipp Otto
Priorität	hoch
Kritikalität	hoch
Verantwortlicher	Philipp Otto
Kurzbeschreibung	Mit diesem Test soll die fehlerfreie Funktion der Kabel getestet werden
Auslösendes Ereignis	Eingabe Konsole: „sudo ./RecordMicToRAM“
Akteure	gesamtes Mikrofonarray, Raspberry Pi, Kabel (lang)
Vorbedingung	Das Array ist vollständig mit Kabeln und Raspberry verbunden, Raspberry eingeschaltet
Nachbedingung	Messwerte grafisch auf Plausibilität überprüft
Ergebnis	Plot der Messwerte
Hauptszenario	1. Über die Konsoleneingabe wird das Programm gestartet 2. Das Programm zeichnet ca. 8 Sekunden lang den Mikrofonausgang auf 3. Das Programm erstellt eine .txt-Datei mit allen Messwerten 4. Die .txt-Datei wird mit python-script in korrektes Datenformat gebracht und in .csv konvertiert 5. .csv wird mit Matlab eingelesen und geplottet 6. Plot wird auf Plausibilität überprüft
Alternativszenario	4.b Die .txt-Datei enthält keine Messwerte 4.c Kabelverbindung wird überprüft 6.b Plot weist fehlerhafte Messwerte auf

Tabelle 2: Kabelfunktionstest Nr. 1

Abschnitt	**Inhalt**
Name	Kabelfunktionstest Nr. 2
Autor(en)	Philipp Otto
Priorität	hoch
Kritikalität	hoch
Verantwortlicher	Philipp Otto
Kurzbeschreibung	Mit diesem Test soll die fehlerfreie Funktion der Kabel bei halbiertem SPI-Takt getestet werden
Auslösendes Ereignis	Eingabe Konsole: „sudo ./RecordMicToRAM64“
Akteure	gesamtes Mikrofonarray, Raspberry Pi, Kabel (lang)
Vorbedingung	Das Array ist vollständig mit Kabeln und Raspberry verbunden, Raspberry eingeschaltet
Nachbedingung	Messwerte grafisch auf Plausibilität überprüft
Ergebnis	Plot der Messwerte
Hauptszenario	1. Über die Konsoleneingabe wird das Programm gestartet 2. Das Programm zeichnet ca. 8 Sekunden lang den Mikrofonausgang auf 3. Das Programm erstellt eine .txt-Datei mit allen Messwerten 4. Die .txt-Datei wird mit python-script in korrektes Datenformat gebracht und in .csv konvertiert 5. .csv wird mit Matlab eingelesen und geplottet 6. Plot wird auf Plausibilität überprüft
Alternativszenario	4.b Die .txt-Datei enthält keine Messwerte 4.c Kabelverbindung wird überprüft 6.b Plot weist fehlerhafte Messwerte auf

Tabelle 3: Kabelfunktionstest Nr. 2

Abschnitt	**Inhalt**
Name	Kabelfunktionstest Nr. 2
Autor(en)	Philipp Otto
Priorität	hoch
Kritikalität	hoch
Verantwortlicher	Philipp Otto
Kurzbeschreibung	Mit diesem Test soll die fehlerfreie Funktion der Kabel bei halbiertem SPI-Takt und kürzeren Kabeln getestet werden
Auslösendes Ereignis	Eingabe Konsole: „sudo ./RecordMicToRAM64“
Akteure	gesamtes Mikrofonarray, Raspberry Pi, Kabel (kurz)
Vorbedingung	Das Array ist vollständig mit Kabeln und Raspberry verbunden, Raspberry eingeschaltet
Nachbedingung	Messwerte grafisch auf Plausibilität überprüft
Ergebnis	Plot der Messwerte
Hauptszenario	1. Über die Konsoleneingabe wird das Programm gestartet 2. Das Programm zeichnet ca. 8 Sekunden lang den Mikrofonausgang auf 3. Das Programm erstellt eine .txt-Datei mit allen Messwerten 4. Die .txt-Datei wird mit python-script in korrektes Datenformat gebracht und in .csv konvertiert 5. .csv wird mit Matlab eingelesen und geplottet 6. Plot wird auf Plausibilität überprüft
Alternativszenario	4.b Die .txt-Datei enthält keine Messwerte 4.c Kabelverbindung wird überprüft 6.b Plot weist fehlerhafte Messwerte auf

Tabelle 4: Kabelfunktionstest Nr. 3

Abschnitt	**Inhalt**
Name	Algorithmus-Test (Matlab)
Autor(en)	Philipp Otto
Priorität	hoch
Kritikalität	hoch
Verantwortlicher	Philipp Otto
Kurzbeschreibung	Mit diesem Test soll die fehlerfreie Funktion des Lokalisierungsalgorithmus getestet werden
Auslösendes Ereignis	„sudo ./RecordMicToRAM64“
Akteure	gesamtes Mirkofonarray, Rasberry Pi, Kabel(kurz)
Vorbedingung	Das System ist vollständig montiert, Raspberry eingeschaltet
Nachbedingung	Lokalisierungsergebnisse grafisch auf Plausibilität überprüft.
Ergebnis	Plot der Lokalisierungsergebnisse
Hauptszenario	1. Über die Konsoleneingabe wird das Programm gestartet 2. Das Programm zeichnet ca. 8 Sekunden lang den Mikrofonausgang auf 3. Das Programm erstellt eine .txt-Datei mit allen Messwerten 4. Die .txt-Datei wird mit python Skript in korrektes Datenformat gebracht und in .csv konvertiert 5. .csv wird in Matlab eingelesen 6. Lokalisierung wird mithilfe des Skripts „testDelayAndSum.m“ berechnet und geplottet 7. Plot wird auf Plausibilität überprüft
Alternativszenario	7.b Lokalisierung entspricht nicht dem erwarteten Winkelbereich 6.c Lokalisierung ist uneindeutig 8. Messung wird in anderem Umfeld wiederholt

Tabelle 5: Algorithmus-Test (Matlab)

Abschnitt	**Inhalt**
Name	Gesamt-Funktionstest
Autor(en)	Philipp Otto
Priorität	hoch
Kritikalität	hoch
Verantwortlicher	Philipp Otto
Kurzbeschreibung	Mit diesem Test soll die fehlerfreie Funktion des gesamten Programms getestet werden
Auslösendes Ereignis	Eingabe Konsole: „sudo ./mic_handler_pipe64“, Start des Programms über Netbeans
Akteure	gesamtes Mirkofonarray, Rasberry Pi, Kabel(kurz)
Vorbedingung	Das System ist vollständig montiert, Raspberry eingeschaltet
Nachbedingung	Lokalisierungsergebnisse grafisch auf Plausibilität überprüft.
Ergebnis	Bearbeitetes .png-Datei
Hauptszenario	1. Über „run“ wird das Hauptprogramm in der IDE gestartet 2. 4 Sekunden warten 3. Mit „sudo ./mic_handler_pipe64“ wird die pipe geöffnet 4. Geräuschevent wird erzeugt 5. Event wird erkannt und auf Konsole angezeigt 6. Wenn das Event im Bildbereich erzeugt wurde, wird ein Bild aufgezeichnet 7. Bild wird bearbeitet und abgespeichert 8. Bild wird auf Zweitgerät geladen 9. Bild wird auf Plausibilität überprüft
Alternativszenario	5.b Event wird nicht korrekt erkannt 5.c Pegelerkennung wird angepasst 5.d Event wird wiederholt 6.b Event wurde im Bildbereich erzeugt aber nicht dort lokalisiert 6.c Event wird wiederholt 9.b Bearbeitetes Bild entspricht nicht den Erwartungen 9.c Lokalisierungsparameter (Buffer wird angepasst) 9.d Event wird wiederholt

Tabelle 6: Gesamt-Funktionstest

6 Tests

Das kontinuierliche Testen und Bewerten aller Einzelkomponenten und Funktionen des Systems ist ein elementarer Entwicklungsbestandteil und teilt den Entwicklungsprozess in unterschiedliche Abschnitte.
Leider hat die aktuell herrschende Covid-19 Pandemie die Erstellung und Durchführung aussagekräftiger Tests erschwert.

6.1 Mikrofontest

Noch vor der eigentlichen Entwicklungsarbeit wurden die zur Verfügung gestellten Einzelkomponenten auf ihre Funktion überprüft. Hierzu wurde der Raspberry Pi mit einem Betriebssystem versehen (Raspberry OS 10) und mit der notwendigen c-Bibliothek versehen BCM 2835 V1.68. Über einfache Jumperkabel wurde jeweils ein zu testendes Mikrofon und ein Abschlussmikrofon an den Raspberry angeschlossen und ein Test durchgeführt. (**??**).

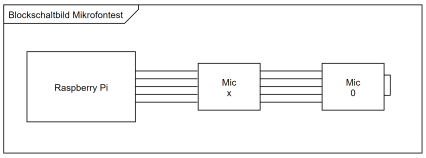

Abbildung 7: Blockbild Mikro Test

Mit dem Skript „RecordMicToRAM“ wurden die Daten des Mikrofonpaar eingelesen und auf ihre Vollständigkeit überprüft. Es wird eine .txt-Datei erzeugt. In dieser Datei sind 7 Spalten und für das entsprechende Mikrofon tauchen bei erfolgreichem Durchlauf Messwerte auf. So konnte die korrekte Funktion der einzelnen Mikrofone garantiert werden.

6.2 Hardware Test

6.3 Kabelfunktionstest

In einem ersten Entwicklungsschritt wurde das gesamte Mikrofonarray mit großzügig dimensionierten Verbindungskabeln (Kabellänge ca. 50 cm) aufgebaut und in Abständen von ungefähr 20 cm zueinander platziert.

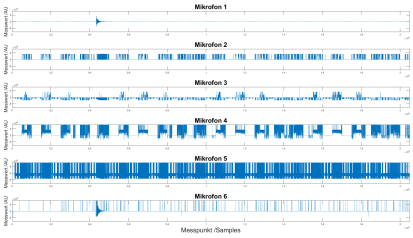

Abbildung 8: Kabelfunktionstest 1

Die dargestellte Messung wurde über einen Zeitraum von acht Sekunden durchgeführt mit einem Händeklatschen nach ungefähr zwei Sekunden.
Die Erwartungshaltung an die Ergebnisse war ein sauberer Ausschlag der Messwerte bei ungefähr einem Viertel der vergangenen Testzeit mit einem anschließenden Abklingen auf den Umgebungspegel.

Lediglich die Aufzeichnungen aus Mikrofon 1 entsprachen den Erwartungen (Ausschlag bei ca. 40.000 Samples), während die restlichen Mikrofone deutliches Rauschen aufwiesen. In der Aufzeichnung von Mikrofon 3 war selbst das Event (erwartet bei ca. 40.000 Samples) nicht zu erkennen, während auf den anderen Kanälen noch Andeutungen eines stärker verrauschten Bereiches zu erkennen waren. Dieses Verhalten hätte seine Ursache beim Raspberry Pi haben können. Dieser muss für die Taktung der Mikrofone einen SPI-Takt über das gesamte Array treiben können. Mit zunehmender Kabellänge nimmt die Taktsicherheit ab.
Durch ein Fachgespräch mit Max Weltz schlossen wir auf folgende mögliche Ursachen:

- Kabel defekt
- SPI-Takt zu hoch
- Kabel zu lang

Um eine qualitative Ursachensuche vornehmen zu können, beschlossen wir die aufgezählten Ursachen separat zu Untersuchen.

6.4 Kabelfunktionstest 2

In einer zweiten Iteration wurde das System umkonfiguriert, sodass jetzt mit einem SPI-Takt von 3,9 MHz und nicht wie ursprünglich mit 7,8 MHz gearbeitet wird.

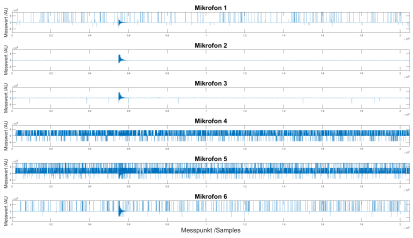

Abbildung 9: Kabelfunktionstest 2

Für diese Konfiguration wurde die gleiche Testmethode verwendet und es ist zu erkennen, dass die Messungen nun deutlich weniger verrauscht waren. Dennoch sind immer wieder bit-shift Fehler zu erkennen, die das Messergebnis zu bestimmten Schwellenwerten springen lässt. Die Messungen waren somit immer noch zu stark verrauscht und für eine Verwendung unbrauchbar.
Eine reine Reduzierung des verwendeten SPI-Takts hat das Problem also noch nicht vollständig gelöst.

6.5 Kabelfunktionstest 3

In der nächsten Anpassung wurde die Kabellänge der Mikrofonkabel auf nun ca. 25 cm reduziert mit der Erwartung, dass die Taktsicherheit über die gesamte Kabellänge steigen sollte.
Die nun durchgeführte Messung wies das gewünschte, rauschfreie Verhalten der Messwerte auf allen Kanälen auf. Die Kabel wurden somit in Bezug auf die Datenübertragung hinreichend auf die Funktion überprüft.

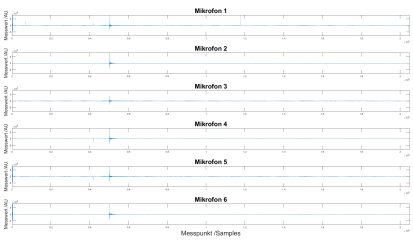

Abbildung 10: Kabelfunktionstest 3

7 Bedienungsanleitung

Zunächst müssen sich der „Raspberry Pi“ und das Zweitgerät in einem gemeinsamen Netzwerk befinden. Dies kann über den Access-Point, den der „Raspberry Pi“ hostet stattfinden oder über beliebige umgebende Netzwerke auf das beide Geräte Zugriff haben. Der Name des AP lautet „wifipi“ und es wird das Passwort „RaspberryWiFi“ verwendet.
Befinden sich beide Geräte im Netzwerk, so kann eine SSH-Verbindung hergestellt werden. Besonders geeignet hierfür ist das Tool „PuTTY“, da es für eine Vielzahl von Betriebssystemen verfügbar ist.
Alternativ dazu kann auch ein Remote Zugriff mit grafischem Desktop wie „VNC Viewer“ verwendet werden. Dieser belastet allerding den Arbeitsspeicher des „Raspberry Pi“ stark, sodass dieser nicht für den laufenden Betrieb des Systems empfohlen wird.
Besteht eine gültige SSH-Verbindung so gibt es nun zwei Betriebsvarianten.
Die Hauptfunktion unseres Systems wird mithilfe von 2 Programmen realisiert. Das eine Programm definiert die SPI-Schnittstelle und liest die Mikrofondaten ein. Diese werden dann über eine Pipeline an das Hauptprogramm gesendet. Für den Entwicklungsprozess wurde das Hauptprogramm über einen Remote Zugriff in der „Netbeans IDE“ entwickelt. So können sämtliche Änderungen am Programm wie etwa die Kalibrierung der Geräuschempfindlichkeit direkt über das zusätzliche Endgerät stattfinden. Diese Empfindlichkeit muss je nach Umgebungslautstärke oder des zu messenden Geräusches eingestellt werden. Zu Testzwecken wurde das System z.B. so eingestellt, dass es in einem ruhigen Umfeld ein Klatschen erkennt. Für die Inbetriebnahme ist es zwingend notwendig, dass zuerst das Hauptprogramm in der IDE ausgeführt wird. Dieses benötigt dann mindestens vier Sekunden, um die Kamera und die Pipeline zu stabilisieren. Im Anschluss daran kann das Skript „mic_handler_pipe64“ über die Kommandozeile gestartet werden. Die Auswertung der Mikrofondaten beginnt und eventuell auftretende Events werden erkannt und berechnet. Befindet sich das Hauptprogramm nicht im Modus „Debug“ läuft diese Auswertung endlos, bis das Programm händisch terminiert wird. Der Prozess wird korrekt beendet, wenn zuerst das Hauptprogramm in der IDE und anschließend das Skript beendet werden.
In der anderen Variante können alle für die Funktion notwendigen Programme mit dem Skript „script.sh“ gleichzeitig gestartet werden. Das Skript muss sich jedoch im gleichen Verzeichnis wie die von der IDE ausführbar kompilierte Version des Hauptprogrammes und das Skript zum starten der Pipeline. Alle drei Skripte befinden sich im /home- Verzeichnis des „Raspberry Pi“ und könnten z.B. mit dem Befehl „bash script.sh“ ausgeführt werden. Je nach Konfiguration des Hauptprogramms läuft der Prozess nun wieder unendlich, in diesem Fall können aber alle Unterprogramme mit einem „stg+C“ Kommando gleichzeitig beendet werden. Wurden in der IDE Änderungen am Hauptprogramm durchgeführt, so muss die aktualisierte Version des kompilierten Hauptprogramms händisch in das korrekte Verzeichnis kopiert werden.
Für einen einfacheren Zugriff auf den Dateiexplorer um z.B. die kompilierten Programme verschieben zu können, empfiehlt sich die Verwendung von „WinSCP“. Hier kann direkt über SSH auf die Ordnerstruktur des „Raspberry Pi“ zugegriffen werden. So lassen sich Kopier- und Bearbeitungsvorgänge sowie die Sicherung und Sichtung der Bearbeiteten Beweisbilder direkt über

das verwendete, zweite Endgerät durchgeführt werden.

8 Fazit