5 Programmierung

5.0.1 Beweisführung

Da ist sich bei dem von uns entwickeltem System um einen Prototyp für ein nutzbaren TDN handelt, hat eine aussagekräftige Beweisführung der Geräusch-Lokalisierungen eine hohe Priorität. Zum Vorbild genommen wurden dafür herkömmliche „Blitzer“, bei denen ein Beweisfoto aufgenommen wird und Metadaten über die gemessene Geschwindigkeit abgespeichert und übermittelt werden.
Das eigentliche Foto wird mithilfe eines Raspberry Pi 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 67° 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 berechneten Quellkoordinaten des Geräusches grafisch darzustellen. Hierbei wird das Bild in eine vorher definierte Anzahl von Segmenten (maximal 70) 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 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 Date, die zur Lokalisierung des Geräusches verwendet wurden in Tabellenform abgespeichert werden. Da es sich hierbei aber bei einem längeren Betrieb um erhebliche Datenmengen handeln würde, die den internen Speicher des Raspberry 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 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 (s. Figure 1).

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

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, 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 $\tau = arg\ _{t \in \mathbb{R}}\ max\,((f \star g)(t))$ 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.

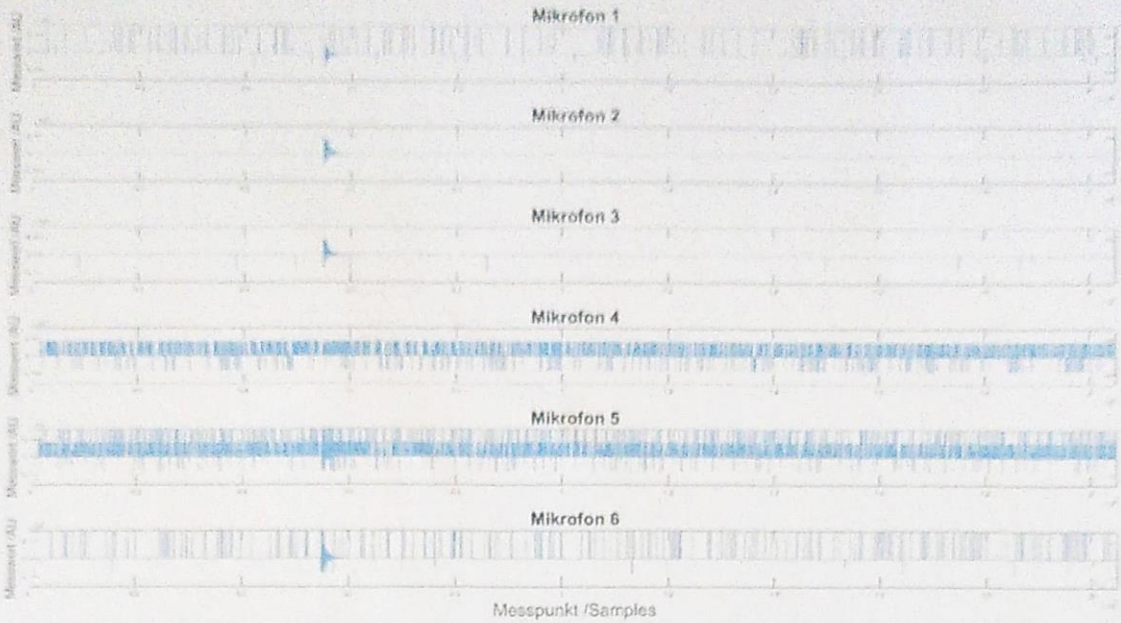

Abbildung 7: Test 2 d

6 Bedienungsanleitung

Da es sich bei dem entwickelten System um einen Prototyp handelt, ist die Bedienung noch nicht benutzerfreundlich gestaltet und bedarf hier einer Einführung.
Das System könnte zwar autonom funktionieren, da aber auf die Verwendung eines Displays verzichtet wurde ist es notwendig ein zweites Endgerät zu verwenden.
Zunächst müssen sich beide Geräte 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 Pipe (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 (Pipesiehe oben) zu stabilisieren. Im Anschluss daran kann das Skript „mic_handler_pipe64“ über die Kommando-

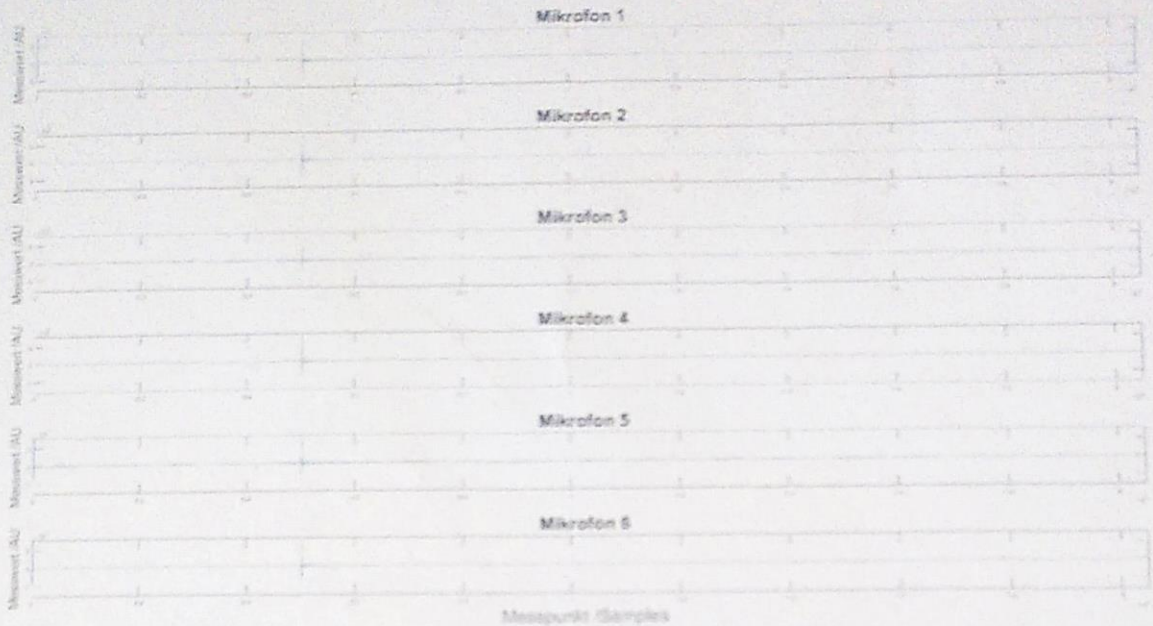

Abbildung 8: Test 3 d

zeile 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 (Pipesiehe oben). 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.

7 Fazit