## 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.

## Hardware Test

## Mikrofonfunktionstest

Noch vor der eigentlichen Entwicklungsarbeit wurden die zur Verfügung gestellten Einzelkomponenten auf ihre Funktion überprüft. Hierzu wurde der Raspberry Pi mit einem Betriebssystem versehen (Respian <Version ergänzen> ) 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 1)

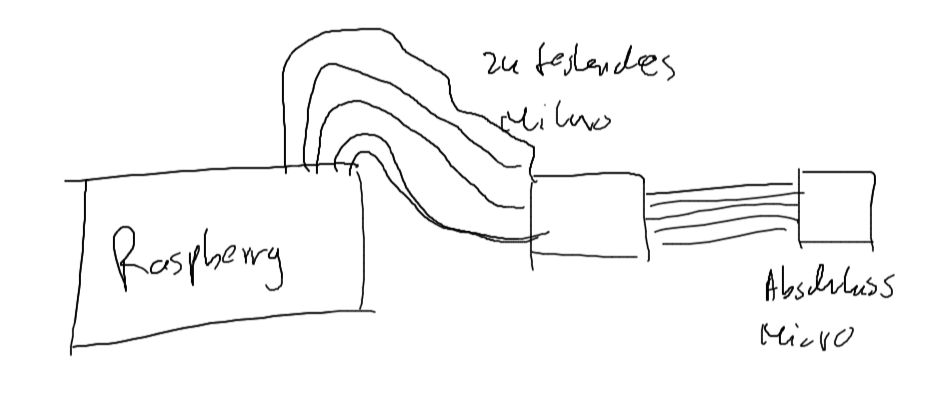


Abbildung 1

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.

## Kabelfunktionstest

In einem ersten Entwicklungsschritt wurde das gesamte Mikrofonarray mit großzügig dimensionierten Verbindungskabeln (Kabellänge ca. 50 cm) aufgebaut und in ungefähren Abständen zueinander platziert. Die daraufhin folgenden Messungen schlossen auf ein Fehlverhalten der Datenübertragung hin.

Abwandlung des Testaufbaus: Längere Kabel, alle Mikrofone angeschlossen

Erwartungshaltung: für jedes Mikrofon entsteht ein Ausschlag auf der Zeitachse

Mögliche Ursachen:

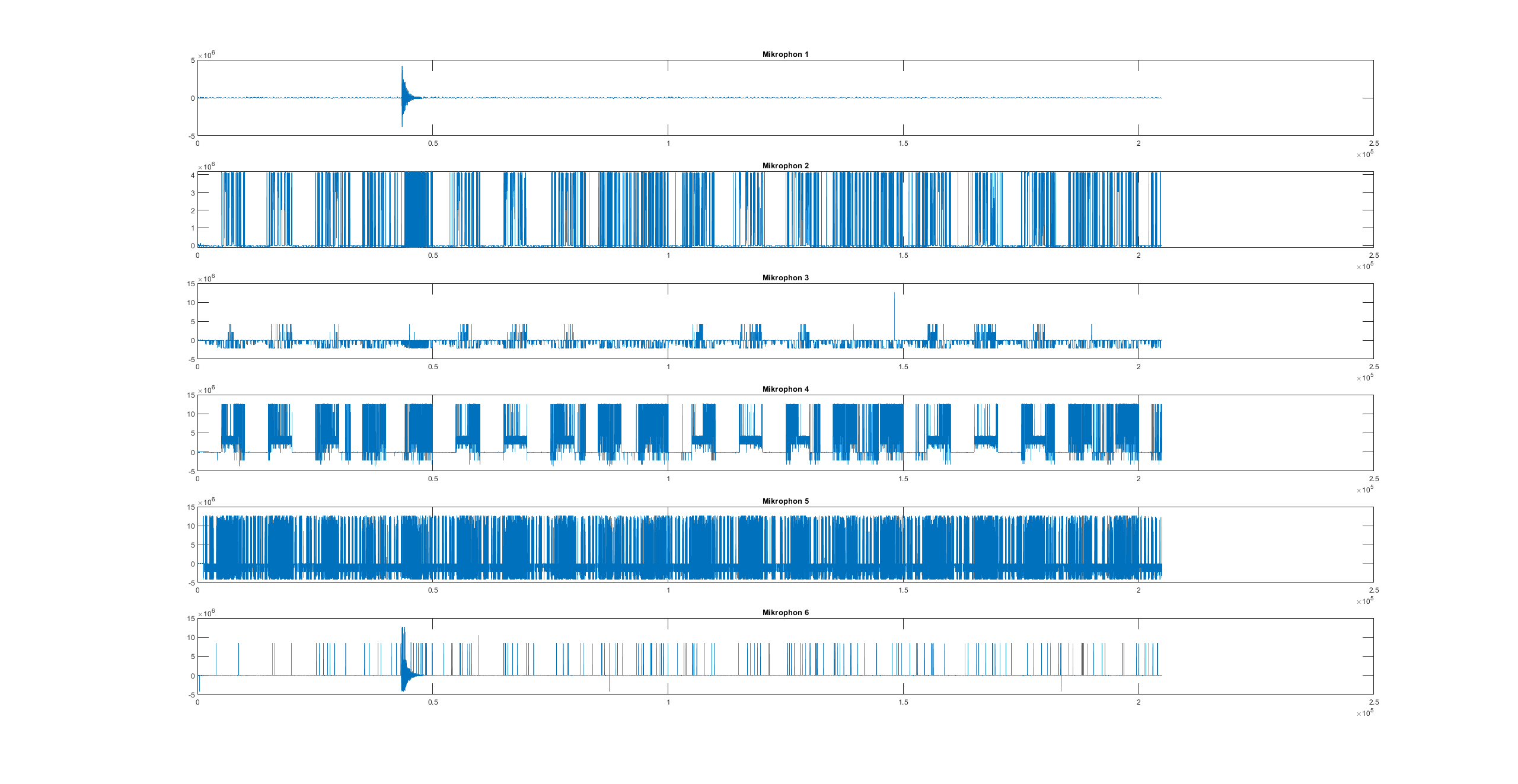
* Kabel defekt
* Taktung zu hoch
* Kabel zu lang

Seperate Untersuchung der einzelnen Ursachen:

Durchgangsprüfung der Kabel:

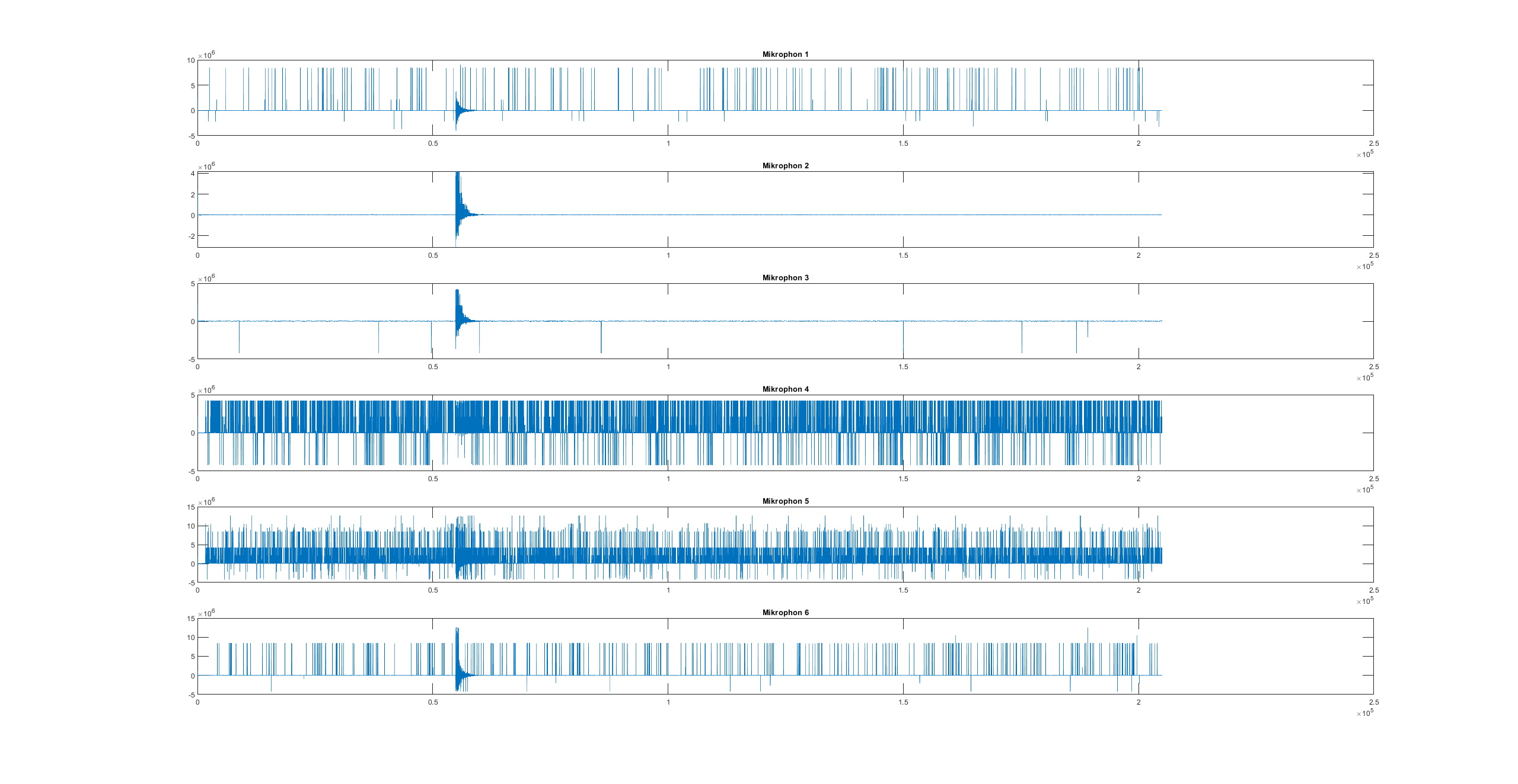
Um eine Kabelprüfung und Kontaktprüfung vorzunehmen, wurde mit einem Multimeter die korrekte Funktionsweise festgestellt.

* Fachgespräch mit Max Weltz

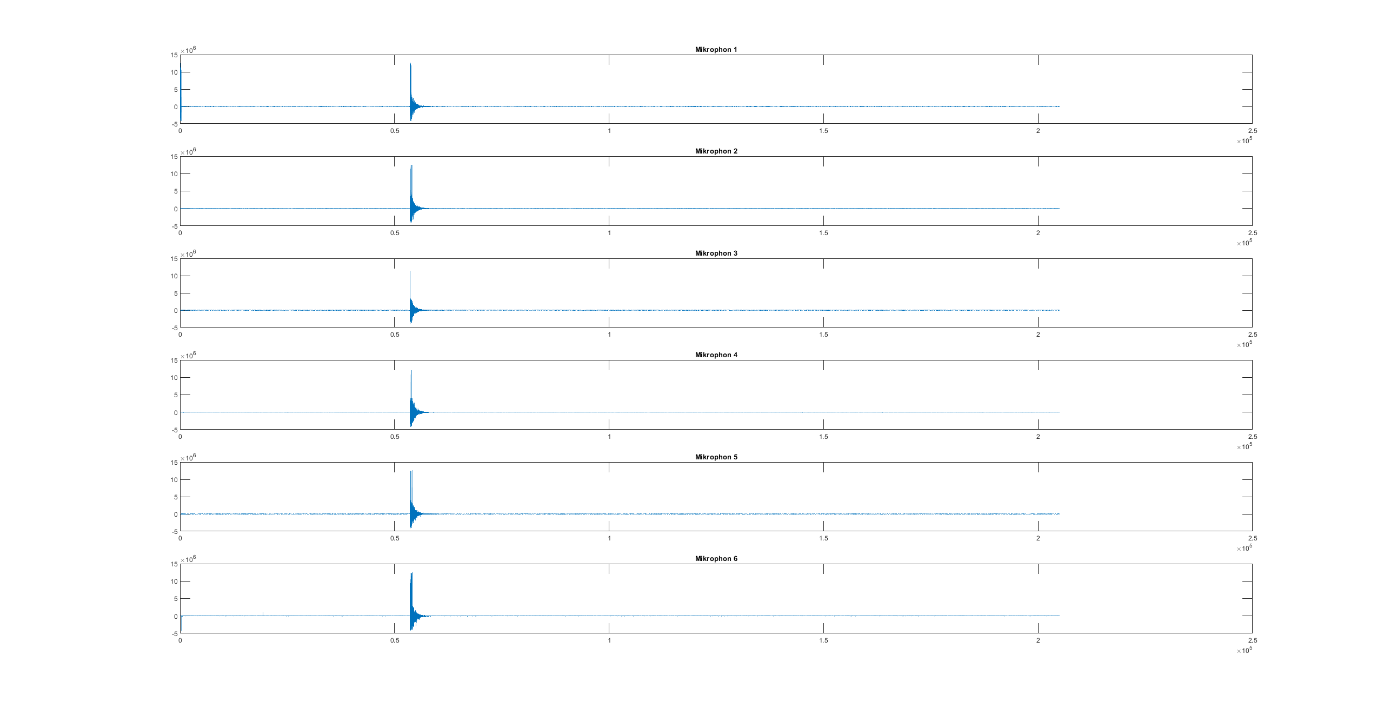


Die dargestellte Messung wurde über einen Zeitraum von acht Sekunden durchgeführt mit einem Händeklatschen nach ungefähr zwei Sekunden.

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. (vgl. Max Studienarbeit?)

In einer zweiten Iteration wurde das System umkonfiguriert, sodass jetzt mit einem SPI-Takt von 3,9MHz und nicht wie ursprünglich mit 7,8125MHz gearbeitet wird. 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.

Fazit der Taktanpassung: Taktanpassung löst das Problem nicht vollständig.

Nächste Anpassung: die Kabellängen der Mikrofonkabel auf nun ca.25 cm reduziert (Erwartung: Taktsicherheit steigt). 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.

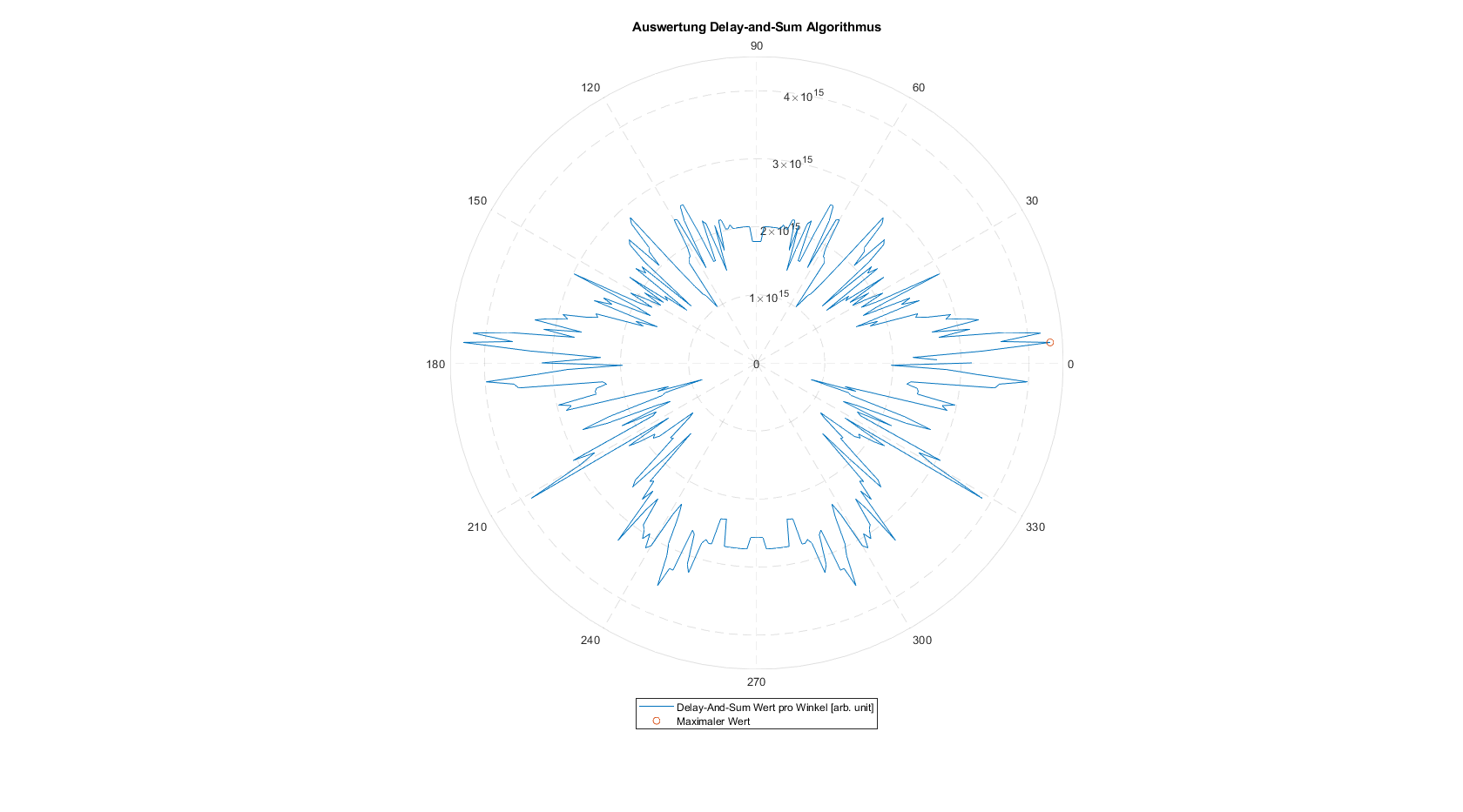
Algorithmustest

Im nächsten Testschritt wurde das Gesamtsystem in einer den Anforderungen entsprechenden Umgebungssituation Situation getestet. Das bedeutet das System wurde auf einem Stativ in fast vollständig aufgebautem Zustand im freien, in einer echoreduzierten Umgebung betrieben.



Zunächst wurde erneut eine Messung über eine Dauer von acht Sekunden durchgeführt und wieder nach ungefähr zwei Sekunden ein Händeklatschen in einer Distanz von ca. 15 m aufgezeichnet.

Die Aufgezeichneten Werte der Messungen, also 205000 Messwerte pro Mikrofon wurden im Anschluss in Matlab mit dem gleichen Algorithmus ausgewertet, der auch Systemintern verwendet wird. Die Lokalisierung ergab folgendes Bild:



Wie zu erwarten ist das Bild der Lokalisierung Achsensymmetrisch entlang der vertikalen, da unser definiertes Koordinatensystem entsprechend rotiert ist. Betrachtet wird in der Auswertung nur der Bereich von 0°-30° und 330°-360°, da dieser Winkelbereich ebenfalls von der Systemkamera abgedeckt wird. Ein Ergebnis der Lokalisierung nahe 0 ° legt also ein Eventursprung frontal des Systems und somit mittig im Bild nahe. Das Ergebnis der Berechnung stimmte ebenfalls mit den Erwartungen überein, da das Geräusch ebenfalls ungefähr frontal vor dem System erzeugt wurde.

Im nächsten Test wurde das System wie bereits im Kapitel „Bedienung“ beschrieben betrieben.

Auch hier wurde das System wieder in einer anforderungsähnlichen Umgebungssituation aufgebaut.



In diesem Test wurde der eigentliche Dauerbetrieb des Systems getestet. Das bedeutet, das System hat lief endlos und durch das Programm wurden fortlaufende Messdaten ausgewertet. Der Versuch hat gezeigt, dass das System den erhöhten Pegel eines auftretenden Events sehr gut erkennen kann. Zwar kam es immer wieder zu variierenden Verzögerungen in der Anzeige auf der Konsole, aber es wurden dennoch aufeinanderfolgende Events korrekt hintereinander aufgelistet.

Der hierzu verwendete Schwellenwert, ab dem das System auslösen soll, musste zunächst auf die neue Umgebungssituation und auf die Lautstärke des Events angepasst werden. Wie im obigen Bild aufgezeigt, konnte sogar der Ursprungsort des Events korrekt lokalisiert werden.

Leider kam es im Lauf der Tests auch immer wieder zu Fehlmessungen.

Obwohl sich der Ursprungsort des Events ungefähr in der Bildmitte befand, wurde der rechte Bildrand als Ursprung lokalisiert. Diese Fehlmessung ist wahrscheinlich auf die Tatsache rückzuführen, dass der Raspberry Pi im laufenden Betrieb des Systems nur eine sehr begrenze Menge an Daten in seinem RAM abspeichern kann. Dadurch waren wir gezwungen die Lokalisierung mit einer deutlich reduzierten Menge an Messwerten durchzuführen.

Leider war es uns nicht möglich ein Testverfahren zu entwickeln, mit dem sich die Genauigkeit der Lokalisierung qualitativ bewerten lässt, um so ein Minimum an benötigten Messdaten ermitteln zu können. Die verwendete Buffergröße wurde lediglich durch Tests ermittelt.

## Zeiten

Für Systeme wie sie in der Geschwindigkeitsüberwachung eingesetzt werden ist die Minimierung der Totzeit sehr wichtig. Sie beschreibt die Zeit, in der das System nicht in der Lage ist, ein neues Event zu erkennen, weil es z.B. noch mit der Datenauswertung oder Datenverarbeitung beschäftigt ist.

Eine Messung dieser Zeiten war nur Systemintern mit Software-timern möglich, die in diesem Fall allerdings hinreichend genau waren.

|  |  |  |
| --- | --- | --- |
| Buffergröße | tBerechnung in s | tGes in s |
| 1024 | 0.03 | 0.27 |
| 5120 | 0.17 | 0.41 |
| 8192 | 0.27 | 0.52 |
| 10240 | 0.34 | 0.57 |

Die Tests haben gezeigt, dass die Berechnungszeit hauptsächlich von der Buffergröße abhängig ist. Zu Testzwecken wurde diese um vielfache der ursprünglichen Anzahl erhöht. Die Berechnungszeit steigt proportional zur Buffergröße an. Vergleicht man nun die Buffergröße mit der Abtastrate der Mikrofone so fällt auf, dass die Berechnungsdauer in Sekunden ungefähr genauso groß ist, wie die Buffergröße in Sekunden.

tGes ergibt sich aus der Berechnungsdauer zuzüglich der Dauer, die das System benötigt das Ergebnis der Rechnung auf seine Gültigkeit im Blickwinkel zu überprüfen, das Bild aufzuzeichnen und das Bild zu bearbeiten. Die Dauer der reinen Bildverarbeitung lässt sich also mit:

berechnen und ergibt für die durchgeführten Messungen ein nahezu konstanter Wert von ca. 0.24 Sekunden. Der Zeitpunkt, zu dem das Bild aufgezeichnet wird, ist jedoch früher im Programmablauf und ist näherungsweise gleichzusetzen mit der Beendigung der Berechnung tBerechnung.

## Fazit

Für eine möglichst genaue Lokalisierung der Eventquelle ist es notwendig der Berechnung möglichst viele Messwerte zur Verfügung zu stellen. Allerdings haben die Tests ergeben, dass es im laufenden Betrieb des Systems nicht möglich ist einen Buffer von mehr als 0.45 Sekunden zu realisieren. Der ursprünglich geplante Buffer von vier Sekunden bringt des Raspberry Pi sogar aufgrund der RAM-Überlastung zum Absturz.

Darüber hinaus verzögert die Berechnungsdauer den Zeitpunkt der Beweisführung linear zur Buffergröße. Selbst wenn es technisch möglich wäre den geplanten Buffer von vier Sekunden verwenden zu können, so wäre das aufgezeichnete Bild dennoch ca. vier Sekunden verzögert. Dieses Verhalten hat zwar keinen Einfluss auf die gegebenen Anforderungen, ein vorbeifahrendes Fahrzeug wäre damit aber unmöglich zu dokumentieren.

Ein Kompromiss aus Lokalisierungsgenauigkeit und Berechnungsdauer ist der Verwendung eines Buffers, der 5120 Messwerte pro Mikrofon enthält. Das entspricht einer Dauer von ca. 0.21 Sekunden. Hierbei ist die vollständige Funktion des Raspberry Pi noch gewährleistet.

Geht man davon aus, dass es sich bei dem Event um einen kurzen Zeitraum wie etwa eine Fehlzündung eines Verbrennungsmotors handelt, so bietet ein Buffer von 0.21 Sekunden dennoch genügend Messwerte, um das Event hinlänglich darstellen zu können.

An einem der Testtage kam es immer wieder zu Fehlerkennungen von Events. Dieses Verhalten konnte auf die Position des Arrays zum Wind rückgeführt werden. An besonders windigen Tagen hat dieser in den Mikrofongehäusen immer wieder Störgeräusche verursacht, die bereits ausreichend waren, die Pegelerkennung auszulösen.

In kommenden Entwicklungen könnten diverse Aspekte verbessert werden. So ist es z.B. möglich die Deckelform der Mikrofongehäuse anzupassen, um diese mit einer Membran abdecken zu können. Eine Abdeckung könnte in Kombination mit einer Hohlraumfüllung der Gehäuse, wie etwa Wolle, die Störgeräusche im Gehäuse reduzieren.

Darüber hinaus ist die Verzögerung der Beweisführung basierend auf der Buffergröße zu adressieren. Eine Möglichkeit dies zu umgehen wäre es das Bild bereits nach der Eventerkennung aufzuzeichnen und zunächst nur im RAM zu speichern. Befindet sich das Event nach der Lokalisierung nicht im Bildbereich, so ist es möglich das im RAM gespeicherte Bild wieder zu löschen und den Speicherplatz für das nächste Bild freizugeben.

Auch der Lokalisierung Algorithmus könnte noch deutlich verbessert werden. Bezieht man die Berechnung nur auf den Blickbereich der Kamera und wertet diesen nicht nach der Rechnung aus wie bisher, so ließe sich die Berechnungszeit reduzieren.

Bisher wird bei der Lokalisierung auch lediglich in der Horizontalen unterschieden. Ergänzt man das Array um eine weitere Dimension, so ließen sich die Events auch in der Vertikalen Lokalisieren. Dies könnte ein großer Vorteil sein, wenn das System z.B. in der Vogelperspektive auf eine Straße runterblickt, wie es häufig auf Autobahnen der Fall ist.

Eine qualitative Bewertungsmethode der Lokalisierungsgenauigkeit wäre ebenfalls notwendig, um die Baugröße des Arrays zu reduzieren. Diese Reduzierung wurden bereits die notwendigen Verzögerungen der Mikrofone berechnet, ohne eine Bewertungsmethode der Lokalisierung ist jedoch nicht möglich ein Minimum festzustellen.