Masterarbeit

Entwicklung und Evaluation eines Datenerfassungssystems für Arraymessungen am Vulkan.

Rick Plescher

Technische Universität Berlin

Technische Universität Berlin

Masterarbeit

Entwicklung und Evaluation eines Datenerfassungssystems für Arraymessungen am Vulkan.

erstellt von

Rick Plescher

[redacted]

Erstgutachter:	Prof. Dr. Stefan Weinzierl, TU Berlin
Zweigutachterin:	Katja Stampka M.Sc., TU Berlin

Thema ausgegeben am:	13.10.2014
Arbeit abzugeben bis zum	11.05.2014
Arbeit abgegeben am:	11.05.2014

Technische Universität Berlin
Fakultät I Geisteswissenschaften
Institut für Sprache und Kommunikation
Fachgebiet Audiokommunikation

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, d. 11.05.2015, Rick Plescher

Inhaltsverzeichnis

Abbildungsverzeichnis

TABELLENVERZEICHNIS

Abkürzungsverzeichnis

AI	Analog Input
API	Application programming interface
BAcIO	The 2014 Broadband Acquisition and Imaging Operation (BAcIO) at Stromboli Volcano (Italy)
BNC	Bayonet Neill Concelman
CCP	Constant Current Power
CMOS-FET	complementary metal-oxide-semiconductor field-effect transistor
DAC	Digital to Analog Converter
DAQ	Data Acquisition bzw.
dBc	Decibel below Carrier
dBFS	Decibel below Full Scale
DFT	diskrete Fouriertransformation
DTOL	Data Translation Open Layer
ENOB	Effective Number of Bits
FFT	Fast Fourier Transformation
FIFO	First In First Out
FPGA	Field Programable Gate Array
GUI	Graphial User Interface
IDFT	Inverse Diskrete Fouriertransformation
IEPE	Integrated Electronics Piezo Electric
INGV	Istituto Nazionale di Geofisica e Vulcanologia
ISTA	Institut für Technische Akustik (der TU Berlin)
LSB	Least Significant Bit
MAMS	Mobile Array Measurement System
PLL	Phased Locked Loop
RAR	(Schall-)Reflexionsarmer Raum
SFDR	Spurios Free Dynamic Distortion
SINAD	Signal to Noise and Distortion Ratio
SH	Sample-and-Hold(-Glied)
SNR	Signal to Noise Ratio
THD	Total Harmonic Distortion
UTC	Universal Time, Coordinated
VERTIGO	Volcanic ash: Field, experimental and numerical investigations of processes during its lifecycle

1

Einleitung

Sinabung, Villarrica, Sakurajima, Shiveluch, Turrialba und der Pico de Fogo sind Namen von Vulkanen mit größeren explosionsartigen Ausbrüchen in den vergangenen Jahren. Einer der wohl bekanntesten Vulkane der letzten Jahre ist der Eyjafjallajökull in Island der mit seinen Eruptionen zwischen April und Mai 2010 zu einer bis dahin nicht gekannten Beeinträchtigung des europäischen Flugverkehrs führte. Grund dafür war die Vulkanasche, welche in einer Höhe zwischen vier bis sechs Kilometer [10] geworfen wurde. Durch die Aschepartikel in der Luft können große Schäden an der Außenhaut der Flugzeuge und der Triebwerke entstehen. Die präventive Einschränkung des europäischen Luftraumes führte zu einem großen wirtschaftlichen Schaden. Die Auswirkungen der Asche beschränken sich nicht nur auf den Luftraum, sondern wirken sich auch auf die in der Nähe eines Vulkans lebenden Menschen aus. Die Infrastruktur kann ebenfalls durch die Asche beschädigt werden und die Gesundheit der Menschen in Mitleidenschaft ziehen.
Die Gefahr die durch die Vulkane entsteht kann nicht verhindert oder kontrolliert werden. Aus diesem Grund ist es wichtig zu verstehen wie die Vulkanasche entsteht und wie sie sich ausbreitet [11]. Die Bedeutung der Gefahren der Vulkanasche anerkennend fördert die Europäische Union seit Januar 2014 das Forschungsprojekt VERTIGO. Ein internationales Netzwerk aus Universitäten und Unternehmen der Luft- und Raumfahrttechnik untersuchen die vielfältigen Aspekte der Vulkanasche. Ziel des Projektes ist es die Entstehung und Auswirkungen der Vulkanasche durch Simulationen, Experimente und Messungen im Feld besser zu verstehen. Die Technische Universität Berlin ist als assoziierter Partner ebenfalls an dem Projekt beteiligt. Die wissenschaftliche Vertretung der TU Berlin erfolgt durch Prof. Dr. Sesterhenn dem Leiter des Fachgebiets für Numerische Fluiddynamik.
Das Forschungsinteresse liegt in der Analyse der vulkanische Eruptionen unter Berücksichtigung der Kenntnisse der Strömungsakustik. Bei einem Vulkanausbruch ist der subjektive Eindruck des dabei entstehenden Geräusche ähnlich wie bei denen die bei einem Flugzeugtriebwerk entstehen. Die Geräusche beim Betrieb eines Flugzeugtriebwerk entstehen durch die turbulenten Vermischung der ausströmende Luft von der Düse und der umliegenden Luft in der Atmosphäre. Diese Art der Strömung wird auch als Freistrahl, turbulenter Freistrahl oder turbulente Strömung bezeichnet. Die Eigenschaften eines Freistrahls sind in den letzten Jahren sehr gut untersucht und dokumentiert worden. Die Kenntnisse über den Freistrahl wurden unter anderem durch die Verwendung von mehrkanaligen Mikrofon-Arrays erreicht, mit denen es Möglichem ist die Verteilung von Schallquellen zu analysieren. Die akustische Analyse von vulkanischen Ausbrüchen mithilfe eines Mikrofonarrays war Ziel einer Forschungsexpedition im Mai 2013 am Stromboli Vulkan, bei der Wissenschaftler der TU Berlin beteiligt waren [12]. Der Vulkan auf der gleichnamigen Insel ist aufgrund seiner regelmäßigen Aktivitäten ein beliebtes Forschungsobjekt für vulkanische Untersuchungen. Mithilfe des Mikrofonarrays sollte das vom Vulkan bei Ausbrüchen erzeugte Schallfeld analysiert werden und

anhand der vorhanden Schallquellstrukturen die Ausströmgeschwindigkeit der Eruption bestimmt werden. Die akustische Analyse eines Vulkans mit einem Mikrofonarray im Nahbereich eines Vulkan stellt einen neuen Untersuchungsansatz in der Vulkanologie dar. Dadurch stellte sich dieses Expedition für die Wissenschaftler der TU Berlin auch als „[...] Überblicksmessung [mit dem] Ziel [war es] den Frequenzbereich der Vulkanausbrüche zu bestimmen, um daraus die Geometrie für ein zukünftiges Array bestimmen zu können [...]“[12]. Erste Ergebnisse zeigten das einige Vulkanausbrüche des Stromboli Eigenschaften eines Freistrahls aufweisen, jedoch verfasst die Autorin auch eine notwendige Optimierung des Array um eine höhere Mikrofon Anzahl und ein größere Array-Geometrie [12, S.857].

1.1. PROBLEMSTELLUNG UND MOTIVATION

Die vorliegende Arbeit entstand als weiterführende Arbeit zu den Messungen von [12]. Im Mai 2014 sollte eine erneute Forschungsexpedition der VERTIGO-Gruppe mit Beteiligung der TU Berlin (Fachgebiet Numerische Fluiddynamik) erfolgen. Aufgrund der Erfahrungen der letzten Expeditionen waren die Schwierigkeiten und Probleme hinsichtlich dem bisher verwendeten Messsystem und der Messumgebung bekannt. Als große Schwierigkeit stellt sich die Erreichbarkeit des Messorts dar. Die Krater des Vulkans liegen auf einer Höhe von etwas 920 m ü.N.N. welche nur zu Fuß mit einem ca. drei stündigen Aufstieg erreicht werden können. Daher kann nur ein begrenztes Gewicht auf dem Rücken transportiert werden. Außerdem muss das gesamte Messsystem Batterie betrieben mehrere Stunden arbeiten. Als Messsystem wurde während der letzte Expedition der Multikanal Analysator OR38 mit 32 Eingangskanälen der Firma OROS verwendet. Aufgrund seines hohen Gewichts von etwa 8 kg ist der Analysator nicht gut für den Aufstieg geeignet. Neben dem hohen Gewicht hat sich als großer Nachteil auch der hohen Leistungsbedarf von maximal 100 W als nachteilig für eine lange Messzeit während des Batteriebetriebs herausgestellt. Ein Betrieb des Messsystems über eine Zeit von zehn Stunden ist so nur mit Akkus mit einer hohen Kapazität von mehr als 1000 Wh möglich ist.
Um während der Expedition BAcIO [13] im Mai 2014 an einem alternativen Messort weitere Arraymessungen durchführen zu können sollte neues Messsystem entwickelt werden. Das neue System soll kompakter und zuverlässiger werden, als das bisherige OROS-Messsystem. Die Entwicklung und Evaluation des neuen Messsystem für die Mikrofonarraymessungen am Stromboli Vulkan im Mai 2014 ist Gegenstand der vorliegenden Arbeit.

1.2. STAND DER FORSCHUNG UND DER TECHNIK

Infraschall-Untersuchungen liefern wichtige Informationen zur Ausbruch-Dynamik und ermöglichen so die Erstellung quantitativer Modelle für Vulkan Ausbrüche. Aktuelle Arbeiten fokussieren sich auf die räumliche und zeitliche Variabilität der Atmosspähre und deren Effekt auf die Infraschall Ausbreitung in Entfernungen zwischen einigen bis 1000 Kilometer. Andere Studien untersuchen den Effekt der komplexen Topografie und der Krater-Morphologie auf das aufgenommene Infraschall-Signal in lokaler und regionaler Entfernung. Die meisten aktuellen Forschungen nehmen eine lineare Ausbreitung von der Quelle aus an. Das ist in so fern kritisch zu sehen, da bereits [14] und spätestens [15] zeigten, dass je nach Ausbruchstyp eine vulkanische Eruption akustisch als Freistrahl angenähert werden können. Ein supersonischer Freistrahl besitzt eine starke Richtwirkung, welche dadurch nicht vollständig erfasst wird.
Vulkane sind stellen eine sehr schwierige Umgebung für Untersuchungen dar. Neben der rauen Umgebung stellt das Fehlen von Straßen und Strom ein herausforderndes Logistikprobleme dar. Außerdem sorgt die dynamische Natur des Vulkan dafür, dass eine geplante Messkampagne aufgrund von geringen Aktivitäten kaum oder keine Daten liefern kann.
Die Untersuchung von Vulkanausbrüchen umfasst eine Multidaten-Analyse verschiedenster Messinstrumente. Zu den klassischen Messinstrumenten gehören z.B. Seismometer um die Stärke der

Erdeben die von einem Vulkan ausgehen zu messen. Zur Analyse der Luft am Vulkan werden häufig Gasspektrometer eingesetzt. Eine weitere Methoden die bei Vulkanbeobachtungen oft zum Einsatz kommt ist die Aufnahme von Wärmebilden mit Infrarotkameras und von Zeitlupenaufnahmen mit Hochgeschwindigkeitskameras. Die Untersuchung der Vulkan-Akustik ist ebenfalls kein junges Forschungsfeld, sondern bereits seit 1976 [14] Gegenstand der Vulkanologie. Woulff schilderte darin bereits die unterschiedlichen Abstrahlcharakteristiken verschiedener Ausbruchsarten von Vulkanen. Dort wird auch gezeigt das ein Großteil der akustischen Energie im nicht hörbaren Infraschallbereich unterhalb von 20 Hz liegt. Einen umfassenden Einblick in die verschiedenen Typen der vulkanische Eruptionen und einen Überblick über die Erkenntnisse des Infraschall findet sich in [16]. Darin wird auch ersichtlich das die bisherigen akustischen Untersuchungen von Vulkanen in einer relativ großen Entfernung von mindestens 15 km und fast ausschließlich in einem Frequenzbereich unterhalb von 20 Hz ausgewertet wurde.
Bisherige akustische Messungen mit einem Array erfolgten bisher mit einer relativ geringen Anzahl an Sensoren. Das Array von [17] bestimmte mit kleiner Apertur den Ort einer Eruption am Stromboli Vulkan. Zum Einsatz kamen Monacor MC2005 Kondensator-Mikrofone (Frequenzbereich $2 - 20\ Hz$). Die Ausbrüche die sie analysierten besaßen je nach Ausbruchsort eine Amplitude von $10\ Pa$ bis $80\ Pa$ und eine Dauer von weniger als 15 Sekunden. Die Daten der Mikrofone wurden mit einer Abtastrate von $54\ Hz$ und einer Genauigkeit von $16\ Bit$ digitalisiert.
Einen Überblick über die aktuellen Untersuchungsmethoden im Bereich der Infraschall von Vulkanen liefert [18]. Moderne geophysikalische Studien integrieren immer mehr Daten mit unterschiedlichen Sensoren z.B. Gas, Temperaturen, Druck, Akustik. Es gibt bereits vielfältige Untersuchungen zu Infraschall von Vulkanen in unterschiedlichen Entfernungen lokal (Abstand kleiner 15 km), regional und global. Es gibt jedoch noch weitere ungeklärte Fragen. So sind noch nicht alle Parameter des akustische Modell eines Vulkans bekannt. Es gibt zum Beispiel keine genaue Kenntnisse über die Beziehungen zwischen den Eruptionstypen und der Schallabstrahlung. Außerdem fehlt es noch an genauen Aussagen zu den atmosphärischen Einflüssen wie Dispersion, Streuung und Dämpfungsverluste.
Die Einführung moderne Analysemethoden mit der Multidaten-Analyse erfolgte spätestens mit [19]. Zu den häufig verwendeten Sensoren für geophysikalische Studien gehört unter anderem die Hochgeschwindigkeitskamera. Mit ihr können Geschwindigkeiten von ausgeworfenen Material und Druckwellen von vulkanischen Eruptionen bestimmt werden [20]. Jedoch ist der Einsatz nur bei guter Sicht möglich. Aus diesem Grund wurden die Beobachtungsmethoden um die akustischen Analyse der Vulkanausbrüchen ergänzt. Mit den Infraschall-Mikrofonen sollen Frühwarnsysteme auch bei schlechter Sicht zuverlässige Ergebnisse liefern.
Um eine Korrelation zwischen einem möglichen Ausbruch und den gemessenen akustischen Signalen herstellen zu können muss die akustische Quellstruktur genau bekannt sein. Aktuelle Forschungen zeigen, das einige Ausbruchstypen von Vulkanen mit einem akustischen Freistrahl verglichen werden können [15]. In diesem Forschungsvorhaben wurde im wesentlichen nur der tieffrequente Schall im Bereich von $0,01\ Hz - 17\ Hz$ analysiert. Die Beschränkung auf den Infraschall-Bereich ist bei vielen Forschungsgruppen zu finden. Im Jahr 2014 wurde in Japan ein umfangreicher Datensatz mit Fokus auf die Vulkan-Akustik erstellt [21]. Bei dieser Mess-Kampagne wurden verschiedene Sensoren zur Messung von Infraschall-Signalen des Vulkans genutzt und zeigen den akutellen Stand der Forschung der Vulkan-Akustik. Es wurden an unterschiedlichen Standorten um den Sakurajima-Vulkan verschiedene Sensor-Arrays aufgebaut. Unter den Sensoren wurden Hyperion IFS-5201 digital Infraschall-Sensoren mit einem Frequenzgang von $0,02\ Hz - 250\ Hz$ im Bereich von $\pm 100\ Pa$ und einer Abtastrate von $500\ Hz$, NCPA Piezo-Keramic-Druckwandler mit 24 Bit A/D-Wandler und GPS-Synchronisation un einem Frequenzgang gerade von $0,02\ Hz - 250\ Hz$ und Dynamik von $\pm\ 1190\ Pa (\approx \pm\ 155\ dB_{SPL})$ und Abtastrate von $500 Hz$. verwendet. Außerdem wurden Allsensor MEMS-Druckwandler (1/2 Zoll) mit einer Dynamik von $\pm 1250 Pa (+/-156\ dB)$ und ei-

ner Abtastrate 200 Hz mit 24 Bit Auflösung eingesetzt. Die Entfernungen zwischen Krater und Array betrug zwischen 250 m bis 14,5 m. Die Auswertung der Daten erfolgte im Bereich $f \leq 20$ Hz. Eine Erkenntnis des Messung was, dass kein starker Zusammenhang zwischen der Amplitude des Infraschall-Signals und der Höhe der Asche-Wolke gibt. Es zeigen sich dennoch deutliche Abhängigkeiten zwischen den Stationen. Es gibt scheinbar aufgrund der topografischen Unterschiede starke Dämpfungen besonders im Bereich der hohen Frequenzen. Es müssen viele Ausbreitungseffekte berücksichtigt werden um auf die Quellprozesse schließen zu können. Demnach sind für die Krater Topografie und Ausbreitungseffekte zu berücksichtigten. Fee zieht in [21] den Schluss,dass die zukünftige Arbeit in der Vulkan-Akustik den Einfluss der Topografie und Veränderung der Quellwerte aufgrund der großen Entfernung untersuchen muss.
Einen anderen Ansatz der Analyse von Vulkanen verfolgt eine Forschungsgruppe bestehend aus des internationale Forschern um das INGV. Ziel der Vulkan-Akustik ist die Verfolgung der akustischen Signale zu ihrer Quelle. Die explosiven vulkanischen Aktivitäten generieren Schall der dem Geräusch eines Freistrahls ähnlich ist. Die Explosionen generieren in Abhängigkeit des Kraterdurchmessers sehr tieffrequente Signale bis hin zum Infraschallbereich bei sehr großen Durchmesser. Der Ansatz von Taddeuci setzt auf die Kombination von Hoch-Geschwindigkeit Kameras, Mikrofon-Aufnahmen und Numerischen Simulationen, um die Eigenschaften des Freistrahls von strombolischen Ausbrüchen heraus zu finden. Ein Freistrahlgeräusch ist nach Taddeuci ein akustisches Feld, welches durch einen Strahl mit hoher Geschwindigkeit aus Gas (mit oder ohne festen oder flüssigen Komponenten) in die umliegende Atmosphäre eintritt und multiple Quellen und damit verbundene charakteristische Geräusche induziert. Taddeuci sieht in anderen Arbeiten zur Vulkan-Akustik zurzeit keine Berücksichtigung der starken Richtwirkung des Freistrahl. Die oft benutzen Beziehungen zwischen gemessener akustischer Leistung und der Geschwindigkeit des ausgestoßenen Gas können daher nicht aussagekräftig sein. Meistens wurde der vulkanische Freistrahl im Bereich von unterhalb von 20 Hz untersucht. Der Freistrahl besitzt jedoch noch weitere hochfrequente Spektralkomponenten mit einer starken Richtwirkung und sind deshalb sehr schwierig in großen Entfernungen zu untersuchen. Der Stromboli-Vulkan hat sekundenlange diskrete Explosionen mit minutenlangen Intervallen von passiver Entgasung. Darum ist er für akustische Studien ein sehr geeignetes Forschungsobjekt, da er große Krater-Durchmesser besitzt.
In Zusammenarbeit mit der TU Berlin [12] wurde eine Messung mit einem 7-kanaligem Mikrofonarray durchgeführt. Während einer Expedition in 2013 sollte untersucht werden, ob sich eine vulkanische Eruption mit den Eigenschaften eines akustischen Überschallfreistrahl vergleichen lassen. In einer anschließenden Expedition in 2014 des INGV[13] fanden weitere Untersuchung statt. Hier wurde eine mulitparametrische Analyse der vulkanische Eruptionen mit zwei Hochgeschwindigkeitskameras, einer Wärmebildkamera, zwei UV-Kameras, zwei breitbandigen Mikrofonen und einem 16-Kanal Mikrofonarray. Die akustische Analyse versucht die Kenntnisse des künstlich erzeugten Freistrahls (z.B. eines Flugzeugs) zu nutzen, um die akustischen Eigenschaften des vulkanischen Freistrahl besser zu beschreiben. Mit akustischen Messungen sollen Computersimulationen des Freistrahls so kalibriert werden um den tatsächlichen Auswurfmechanismus der Eruption in Zukunft genau Vorhersagen zu können. Die Messung des vulkanischen Geräusche erfolgt dabei im hörbaren Bereich bis zu 20 kHz.
Die Datenerfassung muss die Wandlung von akustisch-mechanischen Signalen in elektrische Signale übernehmen. Die Verarbeitung und Speicherung der digitalen Signalen ist ebenso von große Bedeutung. Bei der Datenerfassung handelt es sich um ein komplexes System verschiedener elektronischer und mechanischer Komponenten. Der Stand der Technik ist nicht eindeutig belegt, jedoch kann behauptet werden das moderne Datenerfassungssystem PC-gestützt arbeiten und so flexibel und leistungsfähig gestaltet werden können. Als Sensoren werden immer mehr vorpolarisierte Mikrofone verwendet. Die Gründe dafür werden in Rahmen dieser Arbeit aufgeführt. Genauso wie die überabtastenden $\Delta\Sigma$-Wandler die bei der Analog-Digital-Wandlung verwendet werden.

1.3. ZIEL UND AUFBAU DER ARBEIT

Die vorliegende Arbeit hatte zum Ziel ein Datenerfassungssystem zum Messen von akustischen Signalen des Stromboli Vulkan in Italien zu entwickeln. Es wurden die geeigneten Komponenten des Messsystems zusammengestellt und eine Software zur Erfassung der Messdaten programmiert. Die Arbeit umfasst nicht die Konfiguration der Array-Geometrie oder den Algorithmus des Beamformings.

Das Kapitel 2 beschäftigt sich mit den theoretischen Grundlagen, die teilweise oder ganz, Einfluss auf die mögliche Konfiguration des zu entwickelden Messsystems hatten. Dafür wird unter anderem ein kurzer Einblick in die Entstehung der Geräusche am Vulkan beschrieben.Gefolgt für eine umfassende Behandlung der Datenerfassung. Hier werden die wichtigsten Grundlagen zu allen Komponenten des Datenerfassungssystem dargestellt. Abgeschlossen wird das Grundlagen-Kapitel mit dem Thema der digitalen Signalverarbeitung einschließlich der Analog-Digital-Wandlung und der Arraymesstechnik.

Das vermittelte Wissen befähigt anschließend dazu, dass komplexen Zusammenarbeiten der einzelnen Komponenten (z.B. Mikrofon und A/D-Wandler) nachzuvollziehen und das hier vorgestellt Datenerfassungskonzept zu verstehen. Die Schwerpunkte der Entwicklung lagen vor allem in der geeigneten Dimensionierung der A/D-Wandler und der Entwicklung einer passenden Applikations-Software zur Steuerung der Messung.

Im vierten Kapitel wird versucht ein Urteil über die Funktionsfähigkeit des entwickelten Systems zu treffen. Dafür wird sowohl die Bedienung, als auch elektrische, als auch Parameter aus der Signalverarbeitung (z.B. SNR und THD) zur Einschätzung gemessen und teilweise berechnet. Die technische Datenauswertung wurde dabei für die Labormessungen vorgenommen. Die Feldmessung auf dem Stromboli im Mai 2014 war die eigentliche Zielaufgabe dieser vorliegenden Arbeit. Bis dahin sollte das System fertig sein und stabil arbeiten.

In dem Fazit (Kapitel 5) wird eine kurze Einschätzung darüber gegeben, ob das Ziel erreicht wurde und welche Maßnahmen zur Verbesserung des Datenerfassungssystems in Zukunft getroffen werden müssen.

2

Theoretische Grundlagen

Als Datenerfassungssystem wird eine Messkette bezeichnet die analoge in digitale Signale umwandelt. Das zu untersuchende physikalische Ereignis kann durch ein Vielzahl unterschiedlicher Sensoren erfasst werden. So wird zum Beispiel der Schalldruck durch elektroakustische Luftschallwandler in ein elektrisches (analoges) Signal und anschließend mit der entsprechenden Hardware in digitales Signal umgewandelt, um im Computer gespeichert und ggf. weiter analysiert zu werden. Zum vollständigen Verständnis der komplexen Anforderungen an ein Datenerfassungssystem ist ein bestimmtes Wissen um die spezifischen Eigenschaften der zu untersuchenden Quelle, der Sensoren und der Einzelkomponenten des Datenerfassungssystems notwendig.
In diesem Kapitel wird zunächst ein Überblick über die Vulkan-Akustik gegeben. Anschließend Informationen über die notwendige digitale Signalverarbeitung vermittelt. Abschließend werden die Grundlagen eines Datenerfassungssystem und der Messtechnik von Sensorarrays dargestellt.

2.1. Akustik von Vulkanen

In der Umwelt gibt es eine viele verschiedene Quellen von Geräuschen. Ein Vulkan kann Geräusche erzeugen die oft mit den subjektivem Höreindruck eines Flugzeug-Jets verglichen werden. Um zu verstehen ob Vulkane Jet ähnliche Geräusche erzeugen wird der Blick zunächst auf die Strömungsakustik errichtet, da die Eigenschaften der Flugzeuggeräusche bereits umfangreich untersucht wurden. Dieses Wissen befähigt zur Einschätzung ob und unter welchen Bedingungen ein akustischer Freistrahl durch eine Vulkan-Eruption erzeugt werden kann.
Im Bereich der Strömungsakustik werden Schallquellen in der Regel auf drei wesentliche Quellarten zurückgeführt: Monopol-, Dipol- oder Quadrupolstrahler [22].
Ein Monopol-Strahler (bzw. Strahler 0.Ordnung) liegt z.B. vor, wenn ein Volumen mit einer gewissen Frequenz ausgestoßen und wieder angesaugt wird. Der Großteil der Energie dieser Schallquelle liegt im Bereich der Wiederholfrequenz.
Wird ein Objekt von Luft mit der Geschwindigkeit umströmt wird infolge von Wirbelbildungen am Objekt ein sogenannter Hiebton erzeugt. Der Hiebton besitzt einen Dipol-Charakter. Die Grundfrequenz des Tons ist abhängig von der Strömungsgeschwindigkeit U, dem Durchmesser des Hindernis d und der Strohalzahl St [22, S. 170]

$$f_{Hiebton} = St \cdot \frac{U}{d} \tag{2.1}$$

mit $U = M \cdot c$.
Die dritte Strahlerart von Strömungsgeräuschen in Luft wird durch Quadrupol-Strahler erzeugt. Beispiel für Geräusche die durch dieser Art von Schallquelle entstehen sind die Geräusche eines supersonischen Freistrahls mit einer Machzahl von $Ma \geq 0,7$ [22, S. 173]. Ein akustischer Freistrahl

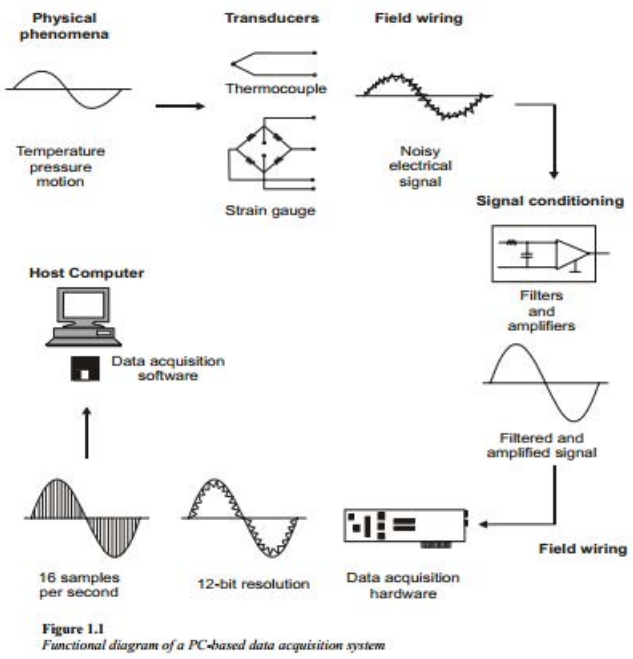
entsteht bei dem Austritt eines Luftstrahl aus einer Öffnung in ein angrenzendes, ruhendes Medium wie etwa Luft. Um den Luftraum herum entsteht eine Grenzzone mit starken Turbulenzen. Bei der Vermischung der beiden Lufträume mit jeweils unterschiedlichen Geschwindigkeiten bildet sich der turbulente Freistrahl aus. In dessen Nahbereich befindet eine Kernzone mit keinen bzw. nur sehr wenigen Wirbeln. An den Rändern des Strahls befindet sich eine turbulenten Mischzone. Mit zunehmenden Abstand von der Öffnung aus dem die strömenden Luft austritt ist der turbulent Freistrahl voll ausgebildet [22, S. 171].
Die Ausbildung eines Freistrahl wird nach [22] von zwei Parametern bestimmt: der Machzahl und der Reynoldzahl. Freistrahlen mit niedriger Reynoldzahl besitzen sehr schmalbandig Geräuschanteile sog. Spalttöne [22]. Die Grundfrequenz wird wie die der Hiebtöne berechnet. Mit steigender Reynoldzahl nimmt der Zerfall der Wirbel an der Grenzschicht der Mischzone zu. Dieser erzeugt ein breitbandiges Geräusch. Der Zerfall von kleinen Wirbeln erzeugt hochfrequente Signalanteile und zerfallende große Wirbel erhöhen den tieffrequenten Signalanteil des Freistrahl. In [23] beschreibt Tam diese Strukturen auch als Small Scale- bzw. Large Scale- Turbulenzen. Er beschreibt auch drei Geräuschanteile des supersonischen Freistrahl: das Turbulent Mixing Noise, broadband Shock-Noise und Screech Ton.
Wie bereits erwähnt kann die ausgeprägte Richtcharakteristik von Freistrahl-Geräuschen mit $Ma < 0,7$ durch einen Kugelstrahler 2.Ordnung (Quadrupol) angenähert werden. Diese Geräusche besitzen ihr Abstrahlmaximum typischer bei einer Strouhal-Zahl von $0,2$ bis $0,3$ [22, S. 171]. Für den Fall einer supersonischen Freistrahls ist die Strouhal-Zahl nur noch von der Machzahl und dem Überwachungswinkel anhängig. Ist der Durchmesser der Austrittsöffnung bekannt kann durch eine Frequenzanalyse des Freistrahl-Geräuschs die Austrittsgeschwindigkeit bestimmt werden.
Die ersten gedanklichen Verknüpfungen bezüglich der Geräusche von künstlich erzeugten Freistrahlen und der Jet von Vulkanen erfolgte durch Woulf und McGetchin [14]. Sie waren die ersten die behaupteten, dass es zwischen der vulkanischen Entgasung und dem tieffrequenten Energieanteil ein Beziehung ähnlich wie der des tieffrequenten Freistrahl-Geräuschs gibt. Eine umfangreiche Analyse des vulkanischen Jets erfolgte unter anderem durch [24]. Er untersuchte die Richtwirkung des vulkanischen Jets, um die Geschwindigkeit und die akustische Leistung des Freistrahl bestimmen zu können.
Da die Eruptionen von Vulkanen sehr unterschiedlich sind gibt es verschiedene Typen zu unterscheiden die als Explosionen, Entgasungen, Tremor oder als Freistrahl bezeichnet werden. Die Dauer eines vulkanischen Ereignisses variiert von sehr kurz (ca. $0,2\ s$) bis sehr lang ($100\ s$). Akustische Untersuchungen zu vulkanischen Ereignisse finden ebenfalls in sehr unterschiedlichen Entfernungen statt. Im weit entfernten Bereich (mehr als $200\ km$ entfernt), im regionalen $10 - -50\ km$ oder im lokalen Bereich innerhalb von $10\ km$. Ausführliche Beschreibungen der genauen Prozesse innerhalb einer Vulkans un der damit verbundenen Geräusche finden sich in [25].

2.2. DATENERFASSUNG

Der Begriff Datenerfassung bezeichnet den Prozess physikalische Ereignisse zu messen in dem sie in elektrische Signale umgewandelt werden. Anschließend werden die analogen Signale in digitale zur Weiterverarbeitung, Analyse und Speicherung im Computer konvertiert. Moderne DAQ-Systeme können nicht nur Messen, sondern auch Teile anderer Syssteme steuern. Ein Datenerfassungssystem (DAQ-System) enthält nach [1] mindestens folgende Komponenten: Sensoren, Messleitungen, eine Signalkonditionierung, DAQ-Hardware, einen PC und eine DAQ-Software. Der funktionale Zusammenhang der einzelnen Komponenten wird in der Abbildung 2.1 dargestellt.
Sensoren messen ein physikalisches Ereignis und geben ein dazugehöriges elektrisches Ausgangssignal aus. Die von Sensoren erzeugten elektrischen Signale sind in der Regel proportional zu den physikalischen Vorgängen. Bei Mikrofonen wird die direkte Proportionalität zwischen physikalischem Ereignis und elektrischen Signal durch den Mikrofon-Übertragungsfaktor angeben. Ein Mi-

krofon mit einem Übertragungsfaktor von z.B. $M = 50\ \frac{mV}{Pa}$ erzeugt bei einem Schalldruck von 1 Pa ein korrespondieren Ausgangsspannung von 50 mV.
Die Ausgangssignale der Sensoren müssen meistens noch in ein Format gewandelt werden, welches akzeptabel für die nachfolgend angeschlossenen DAQ-Hardware ist. Die Anpassung der Signals (Signalkonditionierung) kann die Filterung, Verstärkung, Linearisierung und Isolierung des Eingangssignal beinhalten [1]. Des Weiteren kann die Signalkonditionierung auch eine Versorgungsspannung oder -strom für die Sensoren zur Verfügung stellen, wenn diese benötigt wird.
Für ein Datenerfassungssystem sind die Messkabel ebenfalls eine wichtige Komponente. Sie stellen die einzige physische Verbindung zwischen den Sensoren und der Signalkonditionierung bzw. der Datenerfassung-Hardware dar. Wenn die Entfernung zwischen Sensoren und DAQ-Hardware sehr weit ist müssen die Leitungen entsprechend dimensioniert werden. Sehr lange Messleitungen sind gegen externes Rauschen besonders aufmerksam zu realisieren.
Nach dem das physikalische Ereignis in ein elektrisches Signal umgewandelt und durch die Signalkonditionierung angepasst worden ist, wandelt die DAQ-Hardware das amplituden- und zeitkontinuierliche elektrische Signal in ein digitales amplituden- und zeitdiskretes Signal um. Dieser Vorgang wird allgemein auch als Analog-Digital-Wandlung bezeichnet. Anschließend werden die digitalen Daten zum Computer transferiert.
Moderne DAQ-System sind PC-basiert. Der Host-Computer des PC-basierten DAQ-Systems ist ein wesentlicher Faktor für die Geschwindigkeit des gesamten Datenerfassungssystems. Für Frequenzanalysen und Echtzeit-fähige Anwendungen müssen besondere Anforderungen an das Betriebssystems gestellt werden oder ein entsprechend dimensionierter Buffer (Zwischenspeicher) vorhanden sein.
Ein vollständiges Datenerfassungssystem umfasst nicht nur die dafür notwendige Hardware, sondern enthält auch die dazugehörige Software. Die DAQ-Software stellt die Messergebnisse grafisch dar und kontrolliert das Messsystems. Weiterhin kann die Software auch die Messergebnisse nicht nur darstellen sondern auch analysieren und weiterverarbeiten.

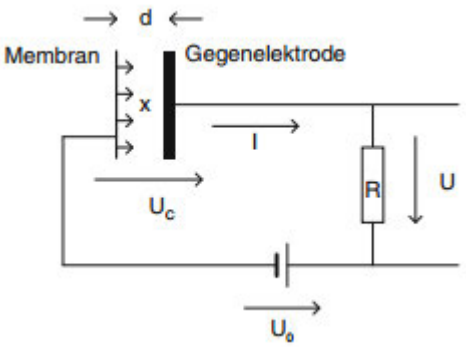
Abbildung 2.1: Funktionsdiagramm eines PC-basierten DAQ-Systems [1].



2.2.1. SENSOREN

Ein Sensor ist ein Gerät das eine bestimmte Form von Energie in eine andere Form umwandelt. In einem Datenerfassungssystems messen Sensoren häufig physikalische Vorgänge und wandeln diese in en korrespondierendes elektrisches Signal um. Sensoren, auch Messumwandler genannt, werden zwischen aktiven und passiven Wandler unterschieden. Aktive Messumwandler wandeln ein nicht elektrisches Eingangssignal in ein elektrisches Ausgangssignal. Die passiven Sensoren ändern in Abhängigkeit des zu untersuchenden Ereignisses den Wert ihres elektrischen Netzwerkes z.B. den Widerstand oder die Kapazität. Zum Betrieb von passiven Sensoren wird eine externe Versorgung in Form eines Spannung oder eines Stroms benötigt.
Sensoren lassen sich nicht nur zwischen aktiv und passiv unterscheiden, sondern auch nach der zu wandelnden physikalischen Größe. Um die Eigenschaften von Sensoren zu beschreiben wird im folgenden nur Bezug auf Sensoren genommen, die den Luftschalldruck in ein elektrisches Signal wandeln. Diese elektroakustischen Luftschallwandler werden auch als Mikrofone bezeichnet.
Es gibt eine zahlreiche Auswahl an Mikrofonen die nach unterschiedlichen Kriterien [S.315 26, Abb. 7.1]. Um die wichtigsten Eigenschaften eines Mikrofons abschätzen zu können sind das Wandlerprinzip, das Arbeitsprinzip und die zu wandelnde Schallfeldgröße vom übergeordneten Interesse. Das Wandlerprinzip bestimmt den elektrischen Aufbau eines Mikrofons. Ein elektrodynamischer Wandler verhält sich elektrisch wie eine Tiefpass 1. Ordnung, wohin gegen ein elektrostatische Wandler sich wie ein Hochpass 1. Ordnung verhält. Neben dem elektrischen Aufbau bestimmt auch der akustisch-mechanische Teil des Mikrofons den Gesamtfrequenzverlauf eines elektroakustischen Luftschallwandlers. Ein Mikrofon kann als Druck- bzw. Druckgradientenempfänger arbeiten und die Membrangeschwindigkeit oder die Membranauslenkung in ein elektrisches Signal umwandeln.
Aus der Kombination Wandlerprinzip, Arbeitsprinzip und Schallfeldgröße wird der Gesamtfrequenzgang des Mikrofons zusammengesetzt. In der akustischen Messtechnik ist das Kondensatormikrofon aufgrund seiner Eigenschaften oft im Einsatz [26, S.323]. Darum und weil Kondensatormikrofone für das in dieser Arbeit vorgestellte Messsytem zum Einsatz kommt, werden die wichtigsten Eigenschaften eines elektroakustischen Luftschallwandler anhand eines elektrostatischen Druck-Auslenkungsempfänger beschrieben. Den interessierten Lesern sei [26] für einen weiterführenden Einblick in die unterschiedlichen Mikrofontypen und deren Eigenschaften empfohlen.
Das mechanisch-elektrische Wandlungsprinzip eines Kondensatormikrofons beruht auf der Modulation einer Gleichspannung durch die Abstandsänderung der Elektroden eines Kondensators. Eine der Elektroden ist eine gefedert aufgehängte Mikrofonmembran, die andere Elektrode ist eine schwere, starr fixierte Gegenelektrode. Durch das einfallende Schallfeld auf der Membran ändert sich der Abstand d der beiden Elektroden zueinander. Die prinzipielle Schaltung eines Kondensatormikrofons wird in der Abbildung 2.2 dargestellt. Für $\omega >> 1/RC_0$, $Q \approx Q_0$ und $\omega_k = \frac{1}{R \cdot C_0}$ gilt das

Abbildung 2.2: Prinzipschaltung eines Kondensatormikrofon [2].



die Ausgangsspannung U

$$\underline{U} = \frac{U_0 \frac{x}{d}}{1 + \frac{\omega_k}{j\omega}} \tag{2.2}$$

direkt proportional zur Membranauslenkung x ist [2, S.360]. Die Gleichspannung U_0 stellt hier die Polarisationsspannung bzw. Vorspannung dar. Je nach Anwendungsgebiet kann sich die Höhe der Polarisationsspannung unterscheiden. In der Tontechnik kommt häufig eine Vorspannung von 48 V zum Einsatz. Extern polarisierte Messmikrofonen verwenden eine Polarisationspannung von 200 V. Es ist auch möglich auf die externe Polarisationsspannung zu verzichten, indem als Gegenelektrode ein dauerpolarisiertes Material (Elektret) verwendet wird [26, S.326]. Die notwendige Signalkonditionierung vorpolarisierter Mikrofone ist günstiger als die der, mit einer externen Spannungsversorgung zu versorgenden, extern polarisierte Mikrofonkapseln.
Bei dem hier vorgestellten Mikrofon handelt es sichm wie zu Beginn erwähnt, um einen elektrostatischen Druck-Auslenkungsempfänger. Der mechanisch-elektrische Aufbau wurde in der Gleichung 2.2 beschrieben. Es handelt sich um einen Hochpass 1. Ordnung mit der Grenzfrequenz ω_k. Der akustisch-mechanische Aufbau reagiert mit der Membranauslenkung (Auslenkungsempfänger) direkt proportional auf den einfallenden Schalldruck (Druckempfänger). Für die anregende Membrankraft gilt somit

$$\underline{F} = \underline{p} \cdot S. \tag{2.3}$$

Das Masse-Feder-System kann durch die folgende Gleichung beschreiben werden

$$F = m \cdot \ddot{x} + r \cdot \dot{x} + s \cdot x\,. \tag{2.4}$$

Für eine harmonische Anregung gilt $F = \Re\left\{\hat{F} \cdot \mathrm{e}^{j\omega t}\right\}$ und $x = \Re\left\{\hat{x} \cdot \mathrm{e}^{j\omega t}\right\}$. Unter Berücksichtigung der komplexen Schreibweise kann die Gleichung für das Feder-Masse-System 2.4 auch mit $\dot{x} = j\omega$ als

$$\underline{F} = \left[m \cdot \left(-\omega^2\right) + r \cdot j\omega + s\right] \cdot \underline{x} \tag{2.5}$$

geschrieben werden. Die mechanisch-akustische Übertragungsfunktion soll das Verhältnis der Membranauslenkung $\underline{x}$ zum einfallenden Schalldruck $\underline{p}$ beschreiben. Das ergibt sich aus der Gleichsetzung der Gleichungen 2.3 und 2.5 und der Umstellung zum gesuchten Verhältnis $\frac{\underline{x}}{\underline{p}}$. Die folgende Gleichung beschreibt die Übertragungsfunktion des Druck-Auslenkungsempfänger.

$$\frac{\underline{x}}{\underline{p}} = \frac{S}{-m \cdot \omega^2 + r \cdot j\omega + s} \tag{2.6}$$

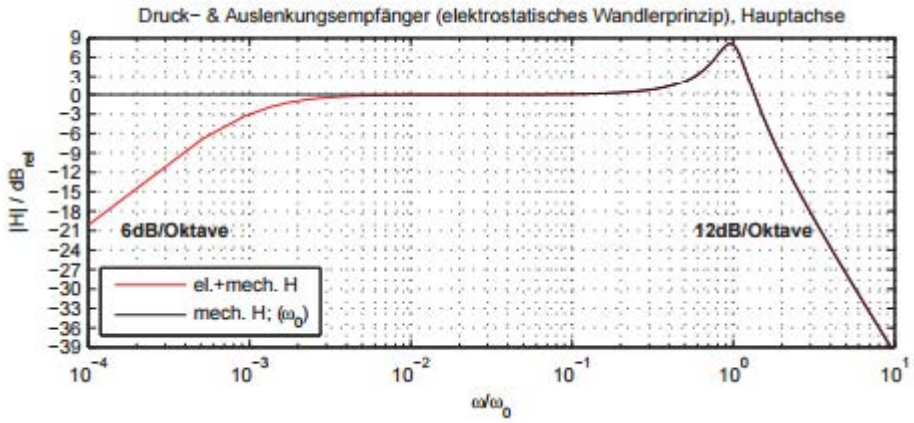
Die Resonanzfrequenz des Teilsystems liegt bei $\omega_0 = \sqrt{\frac{s}{m}}$. Die auf die Resonanzfrequenz normierte Übertragungsfunktion lautet:

$$\frac{\underline{x}}{\underline{p}} = \frac{\frac{S}{s}}{-\frac{\omega^2}{\omega_0^2} + j\eta\frac{\omega}{\omega_0} + 1} \tag{2.7}$$

, wobei für $\eta = \frac{r \cdot \omega_0}{s}$ gilt und auch als Verlustfaktor verzeichnet wird. Der akustisch-mechanische Aufbau des untersuchten Mikrofons verhält sich wie ein Tiefpassfilter 2.Ordnung.
Der Gesamtfrequenzgang des Mikrofons setzt sich jetzt aus dem Frequenzgang des elektrischen Aufbaus (Hochpass 1. Ordnung) und des akustisch-mechanischen Aufbaus (Tiefpass 2.Ordnung) zusammen. Der Frequenzgang ist somit aus 2.2 und 2.7 zu bilden.

$$\frac{\underline{U}}{\underline{p}} = \frac{\frac{S}{s}}{-\frac{\omega^2}{\omega_0^2} + j\eta\frac{\omega}{\omega_0} + 1} \cdot \frac{U_0 \frac{x}{d}}{1 + \frac{\omega_k}{j\omega}} \tag{2.8}$$

Abbildung 2.3: Theoretischer Frequenzverlauf eines elektrostatischen Druck-Auslenkungsempfänger für ebene einfallende Schallwellen [3]



Es zeigt sich somit, dass ein Kondensatormikrofon als Druckempfänger ein Bandpass-Verhalten besitzt (siehe 2.3). Der Frequenzgang steigt mit zunehmender Frequenz bis ω_k mit 6 dB pro Oktave an und besitzt bis zu Resonanzfrequenz $\omega - 0$ einen annähernd konstanten Verlauf. Ab der Resonsanzfrequenz fällt der Freguenzgang um 12 dB pro Oktave. Um einen möglichst breitbandige Frequenzverlauf eines Kondensatormikrofons zu erhalten muss demnach die elektrische Kennfrequenz $omega_k$ möglichst niedrig und die mechanische Resonanzfrequenz ω_0 hoch liegen. Eine hohe mechanische Resonanzfrequnez führt zur Bezeichnung eines hohen abgestimmten Mikrofons.
Die Güte eines Mikrofon-Frequenzgang kann in Klassen eingeteilt werden. Typischerweise werden für Messmikrofone hohe Ansprüche bezüglich des Frequenzgang gemacht werden (Klasse 1 Mikrofone). Neben Frequenzgang wird die Güte eines Mikrofon durch weitere Parameter beschrieben. In der DIN EN 60268-4 [27] sind diese ausführlich beschrieben. Eine wichtige Größe zur Bestimmung des Schalldrucks ist der Übertragungsfaktor M des Mikrofons. Er gibt das Verhältnis zwischen Ausgangsspannung und einfallendem Schalldruck an. Je empfindlicher ein Mikrofon reagiert, desto größer ist der Übertragungsfaktor. In der Regel wird der Übertragungsfaktor bei einer bestimmte Frequenz z.B. 1 kHz gemessen. Um den Übertragungsfaktor im Feld zu bestimmen wird ein akustischer Kalibrator verwendet. Diese Referenz-Schallquelle erzeugt einen definierten Schalldruckpegel i.d.R 94 dB bei $f = 1\ kHz$. Aus der gemessenen Ausgangsspannung kann der Übertragungsfaktor bestimmt werden. Dieser Vorgang wird auch als Kalibrierung bezeichnet. Dabei ist zwischen der Werkskalibrierung und der Feldkalibrierung zu unterscheiden. Die Werkskalibrierung bestimmt dem Mikrofonübertragungsfaktor als Funktion über der Frequenz, während bei der Feldkalibrierung der Übertragungsfaktor bei 1 kHz[1] überprüft und eventuell, aufgrund der angeschlossenen Leitungen und der vorhandenen Umweltbedingungen, angepasst wird. Der Kalibrierpegel kann bei manchen Kalibratoren auf 114 $dBre20\mu Pa$ zu Kalibrierung in einer sehr Fremdgeräusch belasteten Umgebung eingestellt werden.
Ein weiterer Parameter der in der DIN EN 60268-4 aufgeführt wird ist die Richtwirkung. Sie beschreibt das Verhältnis der Empfindlichkeit eines Mikrofons auf der Hauptachse zu den Schalleinfallswinkel. Die Richtwirkung ist frequenzabhängig und nimmt mit zunehmender Frequenz zu. Bei tiefen Frequenzen besitzt ein Druckempfänger eine Kugelcharakteristik. Das bedeutet der gemessene Schalldruck wird unabhängig von der Einfallsrichtung gemessen. Sobald die Wellenlänge des einfallenden Schallfelds in der Größenordnung des Membrandurchmesser kommt stellt die Mem-

[1] Die Kalibrierfrequenz beträgt 1 kHz, weil alle internationalen Bewertungsfilter (A, B, C, D und linear) dasselbe Ergebnis bei dieser Frequenz wiedergeben.

bran ein Hindernis dar und es kommt zu Reflexions- und Beugungseffekten an der Mikrofonkapsel. Dadurch verringert sich die Richtwirkung zu hohen Frequenzen.
Für die Auswahl eines geeigneten Schallwandler für ein Messsystems ist neben dem Frequenzgang und der Empfindlichkeit auch der Dynamikumfang des Mikrofons ein wichtiges Entscheidungskriterium. Dieser gibt im allgemeinen die Differenz aus Grenzschalldruckpegel und Ersatzschalldruckpegel an. Der Grenzschalldruckpegel gibt den maximalen Schalldruckpegel an bei dem eine maximale vordefinierte Grenze für nichtlineare Verzerrungen erreicht wird. Der Ersatzschalldruckpegel bzw. „äquivalente Nennschalldruckpegel der Eigenstörspannung“[27] beschreibt das logarithmische Verhältnis von Geräuschspannung zum Übertragungsfaktor gemessen bei 1 kHz bezogen auf einen Referenzschalldruck von 20 μPa. Die Geräuschspannung kann in der Frequenz bewertet sein z.B. mit dem A-Bewertungsfilter. Die bisherigen Beschreibungen bezogen sich auf ein Mikrofon welches als elektrostatischer Druck-Auslenkungsempfänger arbeitet. Der Begriff Mikrofon"beschreibt im Allgemeinen ein aus mehreren Komponenten bestehendes System bestehend. In der Regel besteht ein Mikrofon aus einer Mikrofonkapsel und einem Mikrofonvorverstärker, der wiederum eine Kombination aus einem Impedanzwandler, einem Filter (optional) und einer Ausgangsstufe [26, S. 370] darstellt. Der Impedanzwandler passt den hochohmigen Ausgang der Mikrofonkapsel an einen niederohmigen Eingang an. Die Ausgangsstufe passt das Signal an den gewünschten Spannungspegelbereich und die Ausgangsimpedanz an.
Zusammenfassend betrachtet müssen bei der Auswahl eines geeigneten Mikrofons im wesentlichen der Frequenzbereich, der Dynamikumfang und die Impedanz berücksichtigt werden. Daraus lässt sich das geeignete Wandler-Prinzip des Sensors und die zu messende Größe ableiten. Moderne Mikrofone werden vorpolarisierte hergestellt, welche wiederum eine zusätzlich zu berücksichtigende Signalkonditionierung benötigen.

2.2.2. SIGNALKONDITIONIERUNG

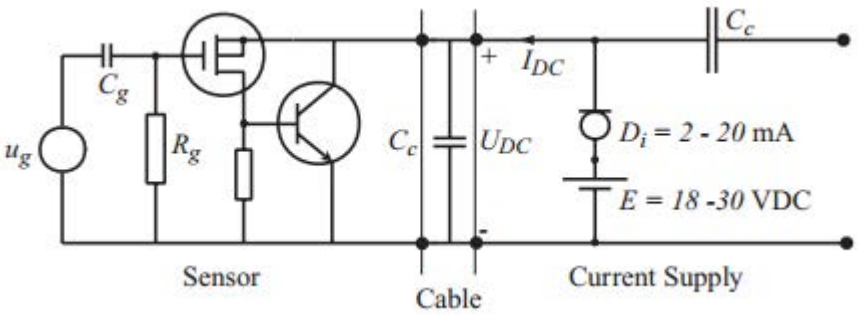
Die prinzipielle Aufgaben der Signalkonditionierung umfassen die Filterung, Verstärkung, Linearisierung, Isolierung und Speisung des Eingangssignals, um es für die anschließende Analog-Digital-Wandlung aufzubereiten [1].
Die Filterung des Eingangsignals ist notwendig, wenn das Signal in einer Umgebung mit hohem Fremdgeräuschanteil aufgenommen wird. So kann ein Tiefpass-Filter hochfrequentes Rauschen reduzieren oder ein Hochpass-Filter elektrisches Störkomponenten z.B. bei 50 Hz entfernen. Allgemein sollten alle Frequenzen außerhalb des zu untersuchenden Messbereichs mit einem Filter gedämpft werden.
Um eine hohe Qualität des Datenerfassung zu erreichen sollte der Spannungsbereich Ausgangssignale der Sensoren möglichst gleich dem Eingangsbereich des nachfolgenden A/D-Wandlers sein. Für Sensoren mit einer geringen Empfindlichkeit kann deshalb eine Verstärkung des Ausgangssignal notwendig sein, um eine hohe Auflösung der A/D-Wandlung zu bekommen. Bei entsprechend dimensionierte Verstärkung kann die Signal-Rausch-Verhältnis und die Auflösung des Signals verbessert werden.
Eine Linearisierung des Eingangssignal ist nur notwendig, falls eine nichtlineare Beziehung der zu messenden physikalischen Größe und dem elektrischem Ausgangssignal des Wandlers besteht. Das trifft für Mikrofone in der Regel nicht zu, weshalb keine Linearisierung für diese Art von Sensoren notwendig ist.
Mit Isolierung wird die galvanische Trennung bezeichnetet die das nachfolgend angeschlossene Equipment vor elektrostatischen Entladungen oder Spannungsspitzen schützt. Außerdem reduzieren sie das fließen von Kriechströmen. Eine gut dimensioniert Isolierung schützt außerdem vor Masseschleifen und vermindert so den Einfluss der elektrischen Störsignale.
Eine weitere wesentliche Aufgabe der Signalkonditionierung ist Speisung der angeschlossenen Sensoren. Wie bereits im Abschnitt 2.2.1 beschrieben können Kondensatormikrofone unterschieden

werden in Mikrofone mit extern polarisierter Mikrofonkapsel und mit vorpolarisierter Mikrofonkapsel. Bei Messungen mit extern vorpolarisierten Mikrofonen muss typischerweise eine Spannung von 200 V zur Verfügung gestellt werden. Diese Spannungsversorgung stellt nicht nur Anforderungen an die Signalkonditionierung sondern auch an die Messleitungen, weil für die Übertragung des Messsignals und der Speisespannung mehrere Adern benötigt werden. In der Regel werden fünf oder sieben-adrige Leitungen verwendet. Die Verwendung einer externen Polarisationsspannung ist mittlerweile als traditionelle Technik zu bezeichnen. Moderne Messsysteme verwenden die bereits in 2.2.1 erwähnten vorpolarisierten Mikrofonkapseln.

INTEGRATED ELECTRONIC PIEZO ELECTRIC

Im industriellen Bereich sowie in der akustischen Messtechnik kommt der von Kistler in den 1960er Jahren entwickelte IEPE-Standard bei der Signalkonditionierung häufig zum Einsatz [4]. Der Industrie-Standard beschreibt normalerweise den Umgang mit Signalen von piezoelektrischer Sensoren, die einen integrierten Impedanzwandler besitzen. Jedoch wird er auch bei vorpolarisierten Elektret-Mikrofonen (vorpolarisierte Kondensatormikrofone) eingesetzt. Der IEPE-Standard wurde von verschiedenen Messtechnikherstellern umgesetzt und mit Hersteller spezifischen Bezeichnungen versehen z.B. ICP (PCB), CCP (G.R.A.S. Tippkemper), Deltatron (Brüel & Kjær). Alle Hersteller spezifischen Signalkonditionierungen entsprechen dem IEPE-Standard und sind deshalb zu einander kompatibel.
Die eingebaute IEPE-Sensorelektronik in etwa die selben Aufgaben, wie die der Mikrofonvorstärker in konventionellen Mikrofonen. Sie wandeln ein hochohmiges Signal in ein Signal mit geringer Impedanz um. Dadurch sind die Verlustleistungen bei langen Kabellängen der Messleitungen gering. Außerdem kann auf den Einsatz speziell geschirmter störungssarmer Kabel verzichtet werden.
Der prinzipielle Aufbau IEPE-Signalkonditionierung wird in der Abbildung 2.4 dargestellt.

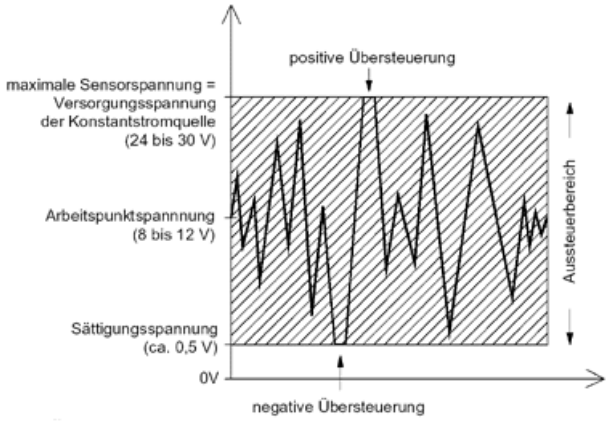
Abbildung 2.4: Prinzipielle Schaltung eines IEPE-Sensors mit einer Konstantstromquelle zur Signalkonditionierung [4].



Der obere CMOS-FET wandelt zusammen mit dem bipolare Transistor die Ausgangsspannung des Sensors mit hoher Impedanz in eine Spannung mit niedriger Impedanz um. Ursprünglich wurde die Elektronikschaltung für piezoelektrische Sensoren entwickelt, weil diese aufgrund der geringen Empfindlichkeit nur geringe Spannungen erzeugen [4, S. 150]. Die in das Mikrofon eingebaute Elektronik wird über den Konstantstrom mit Energie versorgt. Der Versorgungsstrom sowie das Sensorsignal werden über eine Leitung übertragen. Gemäß IEPE-Spezifikation liegt der Konstantstrom zwischen 2 mA bis 20 mA. In Abhängigkeit der Versorgungsspannung der Konstantstromquelle (nach IEPE liegt diese zwischen 18 VDC bis 30VDC) bildet sich eine positive Arbeitspunktspannung beim Sensor auf. Das gemessene Signal vom Sensor (Sensorspannung) wird auf die Gleichspannung am Arbeitspunkt aufmoduliert. Die Sensorspannung kann nicht negativ sein. Zur Erfassung der Daten am Eingang des Datenerfassungsgerät muss der Gleichspannungsanteil des Messsignals herausgefiltert werden. Das geschieht durch den Entkopplungskondensator der dementsprechend dimensioniert ist. Die grafische Darstellung der Arbeitspunktspannung und der

dazugehörigen Grenzen sind in der Abbildung 2.5dargestellt. Der Arbeitsbereich eines IEPE-Sensor

Abbildung 2.5: Darstellung eines aufmodulierten Messsignals eines IEPE-Sensors [5].



wird durch die obere und untere Austeuerungsgrenze bestimmt. Liegen die Ausgangssignale der angeschlossenen Sensoren außerhalb des Arbeitsbereichs der Sensoren kommt es zu einer Über- bzw. Unteraussteuerung des Sensors. Für einen sicheren Betrieb des DAQ-Systems sollte das Messsignal am Eingang des A/D-Wandlers eine untere Grenze von 1 V nicht unterschreiten. Der Minimalwert ist durch die Sättigungsspannung der Impedanzwandlerschaltung definiert. Der untere Aussteuerungsbereich umfasst

$$negative\ range = U_{Bias} - 1\ V. \tag{2.9}$$

Der positive Aussteuerungsbereich eines IEPE-Sensors ist ebenfalls von der Arbeitspunktspannung und zusätzlich von der Speisespannung de Konstantstromquelle abhängig.

$$positive\ range = U_{Source} - U_{Bias} - 1 \tag{2.10}$$

Der gesamte Arbeitsbereich des Sensors ergibt sich aus der Summe von positiven und negativen Aussteuerungsbereich des IEPE-Sensors.
Neben der Dynamik des IEPE-Sensors der durch die Signalkonditionierung bestimmt wird, wird das Frequenzverhalten des Datenerfassungssystem ebenfalls durch das Arbeitsprinzip des IEPE-Standard bestimmt. Für die obere Grenzfrequenz gilt nach [4, S. 152]

$$f_{max} = \frac{I}{2 \cdot \pi \cdot U_{Source} \cdot C_{Cable} \cdot l_{Cable}} \tag{2.11}$$

Demnach kann die maximale Frequenz die mit der IEPE-Konditionierung möglich ist erhöht werden, indem der Konstantstrom I $[A]$ erhöht wird, oder wenn die Versorgungsspannung U_{Source} $[V]$, die angeschlossene Kabelkapazität C_{Cable} $[F/m]$ oder die Länge des angeschlossenen Kabels verringert wird. Die Abbildung 2.6 stellt die Gleichung (2.11) grafisch dar.

Wenn für ein Messsystem ein IEPE-Sensor mit der dazugehörigen Signalkonditionierung verwendet wird ist bei der Analyse der Messsignale zu beachten, das mit zunehmender Frequenz eine Verstärkungsfehler erfolgen kann. Die Abbildung 2.7 zeigt das mit einem Konstantstrom von 4 mA der Verstärkungsfehler mit einer angeschlossenen Kabelkapazität von $20\mu F$ bei eine Frequenz von 20 kHz etwa $1,2(-6,99dB)$ beträgt.

Die obere Grenzfrequenz eines DAQ-Systems mit IEPE-Signalkonditionierung wird somit durch den Anforderung an die Aussteuerbarkeit des Signals und den maximal zulässigen Verstärkungsfehler beschränkt.
Die untere Grenzfrequenz muss, bedingt durch die Entkopplung des Gleichspannungsanteil, größer

Abbildung 2.6: Aussteuerungsgrenze eines IEPE-Sensors in Abhängigkeit der angeschlossenen Kabelkapazität, dem Speisestrom und der zu untersuchenden Frequenz [5].

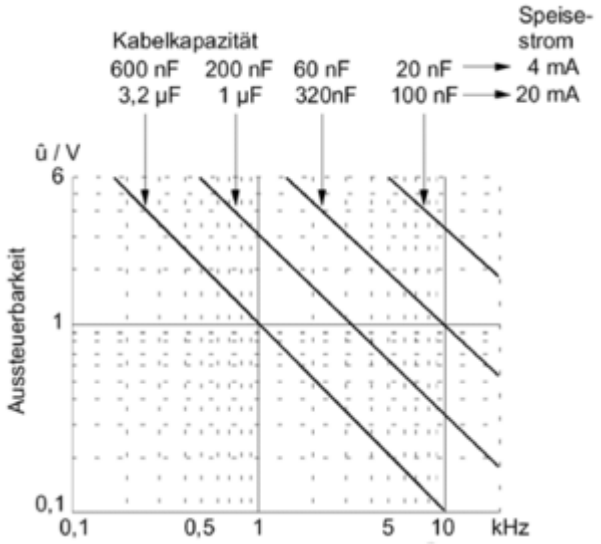
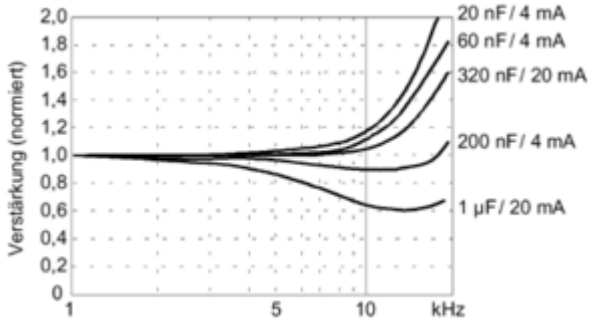


Abbildung 2.7: Verstärkung eines IEPE-Sensors in Abhängigkeit der angeschlossenen Kabelkapazität,des Konstantstroms und der zu untersuchenden Frequenz [5].



0 Hz sein. Die untere technisch realisierbare Grenzfrequenz setzt sich aus der Summe der Zeitkonstanten des Sensors und der Konstantstromquelle zusammen. Diese werden in der Regel durch den Ausgangswiderstand und der Kapazität der jeweiligen Elemente bestimmt und bilden ein Hochpass-Filter.

Es lässt sich festhalten, dass bei der Entwicklung eines Messsystem mit IEPE-Konditionierung neben der richtigen Auswahl der Sensoren auch die richtige Dimensionierung der Signalkonditionierung entscheidend ist, um das gewünschtes Frequenz- und Dynamikverhalten des Datenerfassungssystem zu erhalten.

Die hier vorgestellte IEPE-Konditionierung besitzt eine Reihe von Vorteilen. Die Empfindlichkeit der Sensoren wird nicht durch die Länge und die Art des angeschlossenen Kabel beeinflusst und der niederohmige Ausgang der IEPE-Sensoren erlaubt einen Anschluss von langen Messleitungen. Alternativ ist es auch möglich Kabel zu verwenden, die nicht speziell gegen Störsignale geschützt sind. Zur einfachen Bedingung des Messsystems ist eine Selbsttestfunktion möglich[2]. Dem gegenüber stehen im wesentlichen der Nachteil, dass durch die eingebaute Elektronik im Sensor eine

[2] Bei korrekter Funktionsfähigkeit muss die Spannung am Eingang des DAQ vor der Hochpassfilterung im Bereich der Arbeitspunktspannung liegen. Kann keine Spannung gemessen werden oder ist die Spannung höher als der höchste zulässige Spannungswert kann ein Defekt vorliegen.

zusätzliche interne Rauschquelle vorhanden ist.

2.2.3. MESSKABEL

Unter den Begriff "Messkabel" wird in dieser Arbeit auf die Begriffsbestimmung nach Slavik verwiesen. Nach Slavik[28] sind die Begriffe Leiter, Ader, Leitung und Kabel zu unterscheiden. Ein Kabel ist eine Leitung die zusätzlich durch einen Mantel (z.B. aus PVC) gegenüber Umwelteinflüssen geschützt ist. Eine Leitung besteht im aus mindestens zwei Adern, welche die kleinste Komponente eines Kabels darstellt und aus einen leitenden Material besteht. Eine Ader kann dabei „[...] aus einen einzelnen, starren Draht oder mehreren flexiblen, litzenförmigen Drähten [...]"[28, S.947].
Im vorherigen Abschnitt zur IEPE-Konditionierung wurde bereits ein Vorteil dieses Standards vorgestellt. Durch die Verwendung einer internen Sensorelektronik die nur noch mit einem Konstantstrom versorgt werden muss sinken die Anforderungen die an die Messkabel gestellt werden müssen. Das ist gerade für Mehrkanalsysteme mit langen Kabellängen wichtig, weil so hohe Kosten durch die Kosten pro laufenden Kabelmeter eingespart werden können. Unter Berücksichtigung der IEPE-Konditionierung stellen Koaxialkabel als Messkabel eine preiswerte und qualitativ angemessene Möglichkeit zur Verbindung der Sensoren mit der DAQ-Hardware dar.
Zur Überprüfung der Eignung der Kabel sind die elektrischen und mechanischen Eigenschaften des Kabel zu untersuchen und mit den Anforderungen des jeweiligen Einsatzgebiet zu vergleichen.
Analoge Verbindungen sind anfällig für elektromagnetische Störungen und müssen aufgrund dessen gegen selbige geschützt werden [28, S. 967]. Zur Reduzierung der Anfälligkeit der Messsignal, auf der Übertragungsstrecke, können die analogen Verbindungen mit einer elektronischen Symmetrierung ausgestattet werden. Weiterhin ist die Verwendungen einer abgeschirmten Leitung möglich [28]. Die Abschirmung fängt Störsignale ab und leitet sie gegen die Signalmasse ab. Die Wirkung der Abschirmung ist nach der Meinung von Slavik jedoch nur bei tief- bis mittelfrequenten kapazitiven Störungen wirklich effektiv [28, S. 971].
Eine wesentlich Frage bei der Auswertung von mehrkanaligen Messsystem ist immer die Frage der Latenz der einzelnen Kanäle zueinander. Das ist bei Kabellänge von $\leq$ 100 m jedoch zu vernachlässigen, weil die Ausbreitungsgeschwindigkeit in Kupferleitungen so hoch ist, dass eine Latenz quasi nicht vorhanden ist. Längenunterschieden der einzelnen Messkabel führen daher zu keinem nennenswerten Fehler in der Datenerfassung.

KOAXIALKABEL

Durch die etwas aufwendigere IEPE-Sensorelektronik der entsprechenden Sensoren ist es einfache Koaxialkabel zu verwenden ohne auf hohe Ansprüche zur Messqualität erwarten zu müssen. Die elektrischen und mechanischen Eigenschaften und die, je nach Typ, geringen Kosten pro laufenden Meter sind Grund für einen verbreiteten Einsatz dieser Kabel in der Messtechnik. Mit einem Koaxialkabel können hochfrequente analoge Signal übertragen werden. Es ist ein zweipoliges Kabel mit konzentrisch angeordneten Kreisen. Ein Innenleiter wird in einem konstantem Abstand von einem Außenleiter umgeben. Dieser ist hohl-zylindrisch ausgeführt und schirmt den Innenleiter vor kapazitiven Störsignalen ab. Zwischen dem Innen- und Außenleiter ist ein Dielektrikum angeordnet. Um den Außenleiter ist außerdem noch ein Mantel angebracht um das Messkabel vor Schäden durch Umwelteinwirkungen zu schützen.
In der elektrischen Betrachtung des Koaxialkabels stellt der Außenleiter die Abschirmung (Signalmasse) dar und der Innenleiter die Signalführung (Ader) dar. Je nach Ausführung der Kabel kann zwischen Dielektrikum und den Innen- bzw. Außenleiter noch ein weiteres Material angebracht werden um ein spezielles Rauscharmess Kabel zu erhalten. Außerdem kann durch die Verwendung eines Dielektrikums mit geringem Verlustfaktor die Signaldämpfung des Kabels verringert werden. Eine wichtige Kenngröße von Kabel ist der Wellenwiderstand. Für typische Anwendungen in der akustischen Messtechnik werden Koaxialkabel mit einem Wellenwiderstand von 50 Ω verwendet. Er ist unabhängig von der Leitungslänge und Signalfrequenz. Das elektrische Ersatzschalt-

bild eines Kabels enthält sowohl resistive, als auch kapazitive und induktive Komponenten. Daher muss bei der Entwicklung eines Datenerfassungssystems ebenfalls das frequenzabhängige Verhalten der Messkabel berücksichtigt werden. Der Frequenzgang der Kabel wird durch die Angabe des Induktivitäts- und Kapazitätsbelag gekennzeichnet. Diese geben an wie groß die Induktivität oder Kapazität eines Kabels mit einem Meter Länge ist. Die Gesamtdämpfung des Signals ist somit von der Signalfrequenz und der Gesamtlänge des Kabels abhängig. Der Dämpfungsfaktor beschreibt in de Regel die Dämpfung des Signals bei einer bestimmten Frequenz bezogen auf einen Meter Länge. Zur Abschätzung der Ausbreitungsgeschwindigkeit in der jeweiligen Leitung wird der Verkürzungsfaktor angegeben. Es beschreibt das Verhältnis der Ausbreitungsgeschwindigkeit von Licht in der Luft zu der Ausbreitung der Elektronen in dem Kabel. Zur Verbindung des Koaxialkabel an die Hardware und die Sensoren werden üblicherweise BNC-Steckverbinder verwendet und können für Frequenzen bis maximal 1 - 4 GHz verwendet werden.

2.2.4. HARDWARE

Zur angemessenen Dimensionierung der Hardware eines Datenerfassungssystem sind verschiedene Faktoren zu berücksichtigen. Die wichtigsten Punkte sind die Art und Anzahl der Sensoren, die Position der Sensoren bezogen auf die Datenerfassungshardware, die Art der Signalkonditionierung und die Messumgebung.
Die Hautaufgabe der DAQ-Hardware stellt die Analog-Digital-Wandlung der analogen Signale der Sensoren in digitale Signale zur Weiterverarbeitung, Analyse, Darstellung und Speicherung im Host-PC. Moderne DAQ-Hardware enthält mehrere Komponenten z.B. A/D-Wandler, Takt-Generator, Prozessors, Bus-Systeme um eine hochwertige Signalerfassung zu ermöglichen. Ein sehr wichtige Komponenten ist, wie bereits erwähnt der A/D-Wandler. Dieser muss entsprechend der zu erfassenden Signale dimensioniert sein. Zu den wichtigen Eigenschaften des A/D-Wandler gehören der Dynamik-Bereich, die Amplitudenauflösung und die zeitliche Genauigkeit.
Die Güte der Amplitudenauflösung wird in der Regel mit dem LSB angegeben. Es beschreibt das Verhältnis des Eingangsbereichs (analoge Spannungswerte die ein gültiges digitales Wort erzeugen) zu der Anzahl der Quantisierungsstufen des A/D-Wandlers und stellt die Abstände der einzelnen Quantisierungsstufen zueinander dar. Für bipolare koventionelle A/D-Wandler kann die Auflösungen nach gemäß 2.12 berechnet werden.

$$LSB = q = \frac{|U_{max} - U_{min}|}{2^N} \tag{2.12}$$

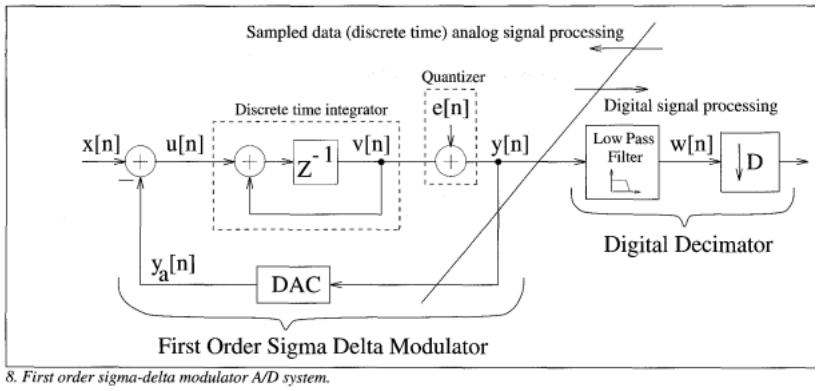
Ein 24 Bit A/D-Wandler mit einem Eingangsbereich von $U_{min} = -10\ V$ bis $U_{max} = 10\ V$ besitzt eine theoretische Amplitudenauflösung von $LSB = 1,19\ \mu V$. Durch die Quantisierung wird ein Fehler gemacht der nicht reversibel ist. Bei einem ideal äußert sich dieser Fehler als sogenannten Quantisierungsrauschen, welches gleichverteilt von $-LSB/2$ bis $+LSB/2$ ist. Die Leistung des Quantisierungrauschens hängt nur von der Auflösung des A/D-Wandlers ab.
Für eine möglichste genaue Umwandlung der analogen Signalen muss der Eingangsbereich des A/D Wandlers mit dem Ausgangsbereich der Sensoren übereinstimmen. Übersteuerungen sind zu verhindern, weil ein A/D-Wandler empfindlich auf solche reagiert [28, S. 967]. Durch eine Übersteuerung werden zusätzliche harmonische Verzerrungsprodukte [3].
In der modernen Messtechnik hat sich die Verwendung von ΔΣ-Wandler als Stand der Technik durchgesetzt. Diese Wandler besitzen ein grundsätzlich anderes Verhalten gegenüber anderen konventionellen A/D-Wandlern [4] und stellt den aktuellen Stand der Technik in der Datenerfassung dar.

[3] Die Verzerrungsprodukte entstehen durch das sogenannte "Clipping„. Dabei wird das Eingangssignal im obere bzw. unteren Spannungsbereich abgeschnitten und es entstehen rechteckähnliche Signalverläufe. Die wiederum enthalten Frequenzanteile bei den ungeradzahligen Vielfachen der Grundfrequenz.

[4] Unter konventionellen Verfahren sind A/D-Wandler mit den Arbeitsprinzipien der sukzessiven Approximation und Parallel-Verfahren zu nennen.

Ein ΔΣ-Wandler besteht aus einem ΔΣ-Modulator und einem digitalen Dezimator genannt (siehe Abbildung 2.8). Der Wandler konvertiert die Daten mit einen sehr hohen Abtastrate und verschiebt dabei einen gewissen Anteil des Quantisierungsrauschen in einen Frequenzbereich, der außerhalb des zu untersuchenden Messbereichs liegt.
Die Verschiebung der Rauschleistung erfolgt aufgrund des Funktionsprinzips des ΔΣ-Modulators

Abbildung 2.8: Aufbau eines ΔΣ-Wandler 1. Ordnung [6]



8. First order sigma-delta modulator A/D system.

(siehe Abbildung 2.8). Der ΔΣ- Wandler verwendet lediglich einen 1-Bit A/D-Wandler. Anschließend wird das quantisierte 1-Bit Signal in einer Rückkopplungsschleife auf das Eingangssignal zurück geführt. Die Differenz des aktuellen Samples des Eingangssignals und dem quantisierten DAC-Ausgangssignals wird auf den Integrator zurückgeführt. Dadurch wird das digitale Signal in der Amplitude “nachgeregelt". Der Modulator arbeitet dabei nicht mit der nominalen Abtastfrequenz, sondern wird um einen bestimmten Faktor überabgetastet betrieben (Oversampling). Die Abtastung des Signals ist dadurch wesentlich höher, als sie mindestens zur Auswertung der maximal darzustellen Frequenz sein sollte. Dadurch kann eine Verbesserung des SNR erreicht werden. Weil das Quantisierungsrauschen unabhängig von der Abtastfrequenz ist und sich breitbandig über die gesamte Signalbandbreite verteilt wird das Quantisierungsrauschen um den Überabtast-Faktor reduziert [29]. Außerdem kann ein weniger steilflankiges Anti-Aliasing-Filter verwendet werden, weil die maximale zu untersuchende Frequenz nicht mehr so dicht an der Nyquist-Frequenz liegt. Der Vorgang der Reduzierung des Quantisierungsrauschens durch eine erhöhte Abtastrate wird als Noiseshaping bezeichnet.
Die Übertragungsfunktion des ΔΣ-Modulators lautet

$$Y(z) = X(z) \cdot H_x(z) + E(z) \cdot H_e(z) \tag{2.13}$$

Die Z-Transformierte des Quantisierungsrauschen wird durch den Term $E(z)$ beschrieben. Unter der Annahme eines idealen DACs und der Übertragungsfunktion des Intergrators mit $H_x(z) = \frac{z^{-1}}{1-z^{-1}}$ lautet die Übertragungsfunktion des Modulators aus (2.13)

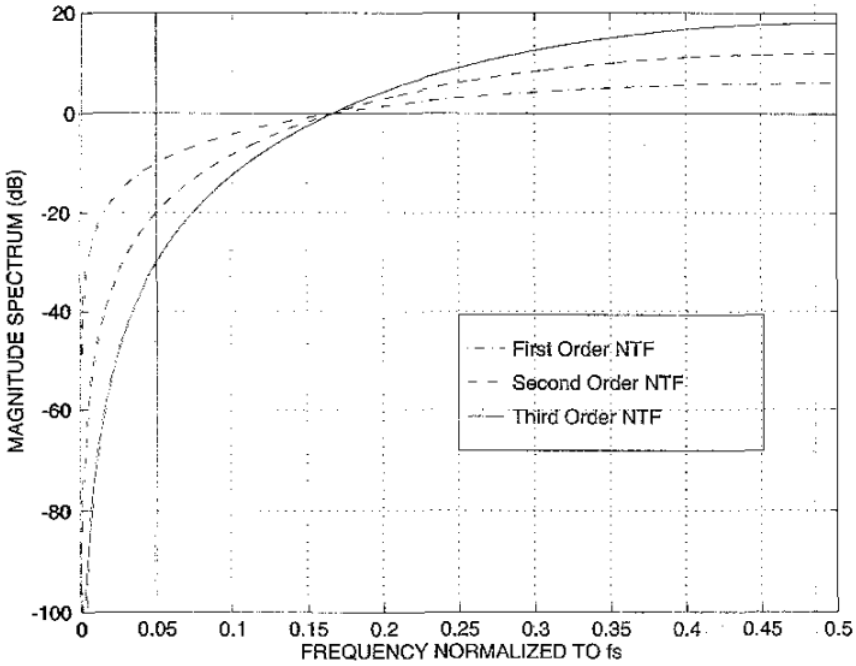
$$Y(z) = X(z) \cdot z^{-1} + E(z) \cdot \left(1 - z^{-1}\right) \tag{2.14}$$

Die Gleichung (2.14) stellt die Signalübertragungsfunktion z^{-1} und die Rauschübertragungsfunktion $1 - z^{-1}$ dar. Für das abgetastete Signal im Zeitbereich gilt für (2.14)

$$y[n] = x[n-1] + e[n] - e[n-1] \tag{2.15}$$

Die Gleichungen (2.15) und (2.14) zeigen, dass das Ausgangssignal des Modulators 1. Ordnung aus der Summe des spektral geformten Quantisierungsrauschen (Noiseshaping 1. Ordnung) und dem,

Abbildung 2.9: Übertragungsfunktion des Noiseshaping 1./2. und 3. Ordnung [6]



um ein Sample verzögertem Eingangssignal besteht. Die spektrale Formung durch die Rauschübertragungsfunktion entspricht dabei einer Hochpassfilterung. Unter Berücksichtigung des Oversamplings-Faktors r gilt $2^r = \frac{f_s}{2 \cdot f_B}$. Nach [6] berechnet sich das Signal zu Rauschverhältnis (SNR) am Ausgang des $\Delta\Sigma$-Modulators dann folgendermaßen

$$SNR = 10 \cdot \log\left(\sigma_x^2\right) - 10 \cdot \log\left(\sigma_e^2\right) - 10 \cdot \log\left(\frac{\pi^2}{3}\right) + 9{,}03 \cdot r. \tag{2.16}$$

Jede Verdopplung des Over-sampling-Faktors verbessert das SNR um 9 dB. Das entspricht einer Verbesserung der Auflösung um $1{,}5\ Bit$.

Die Güte eines A/D-Wandler wird nicht nur durch das SNR wiedergegeben, sondern durch weitere Werte wie die THD, SFDR, SINAD und ENOB. Diese Parameter können mit einer FFT-basierten Analyse ermittelt werden.
Der THD oder Klirrfaktor stellt das Verhältnis des Effektivwertes der Signalamplitude zur der Summe des Effektivwertes der harmonischen Schwingungen dar. Der Klirrfaktor kann in dBc oder dBFS gemessen werden. Durch dBC wird das Verhältnis zur aktuellen Eingangsamplitude berücksichtigt, während durch das Messen in dBFS dass Verhältnis zum gesamten Eingangsbereich berücksichtigt wird. Mit der Ordnung K der Harmonischen Verzerrungen des Eingangssignalfrequenz f_a und der Abstastrate f_s können die Frequenzen der zu erwartenden harmonischen Verzerrungen f_{HD} berechnet werden [30].

$$f_{HD} = \left|\pm K \cdot f_s \pm n \cdot f_a\right| \tag{2.17}$$

$$THD[dB] = 20 \cdot \lg\left(\frac{U_2^2 + U_3^2 + U_4^2 + \ldots + U_n^2}{U_1^2}\right) \tag{2.18}$$

Unter Berücksichtigung der Rauschleistung inklusive der Effektivwerte der harmonischen Verzerrungen kann das SINAD berechnet werden [30].

$$SINAD = 10 \cdot \lg\left(\frac{P_S}{P_D + P_N}\right) \tag{2.19}$$

Das SFDR beschreibt den Abstand der Amplitude des Eingangssignal zur größten Störkomponente im FFT-Analysefenster. Eine in der Praxis relevante und aussagefähige Größe ist die ENOB, die effektive Anzahl der Bits.

$$ENOB = \frac{SINAD - 1,76}{6,02} \tag{2.20}$$

Sie beschreibt im Gegensatz zum theoretisch erreichbaren SNR bei der Angabe der Amplitudenauflösung, welche Auflösung tatsächlich vorhanden ist.

2.2.5. SOFTWARE

Neben der Hardware ist die Software für ein PC-basiertes Messsystems ebenfalls von herausragender Bedeutung. Damit die Software zuverlässig funktioniert muss ein entsprechendes Betriebssystem vorhanden sein, welches die grundlegende Kommunikation zwischen den Systemkomponenten wie z.B. die Festplatte und Bildschirm und der Software zu Verfügung und verwaltet die Ressourcen des Computers.

2.3. DIGITALE SIGNALVERARBEITUNG

In der Regel liegen die zu untersuchenden Signale zu Beginn als analoges Signal vor. Das bedeutet sie enthalten eine wechselnde Amplitude bezogen auf die Zeit mit kontinuierlichen Werte- und Zeitbereich. Die Stärke eines analogen Signals hängt von der Variation der Amplitude, dem Zeitverlauf und den Frequenzanteilen des Signals ab. Generell sind die analogen Signale zwischen DC-Signale und AC-Signale zu unterscheiden. Die analogen DC-Signale sind statische bzw. langsam variierende Signale. Die Information dieses Signaltyps liegt in der Höhe seiner näherungsweise konstanten Amplitude.
Die Information eines analogen **AC!**-Signale liegt nicht in der Höhe der Amplitude zu einem Zeitpunkt, sondern auch über im Zeitverlauf der variierenden Amplitude. Die Form des Signals und die Lage der Signalspitzen liefern die Information zum gesamten Signal.
Ein analoges Signal besitzt einen kontinuierlichen Werte- und Zeitbereich. Durch eine zeitliche Abtastung wird aus dem analogen Signal ein zeitdiskretes und wertekontinuierliches Signal. Die mathematische Beschreibung der idealen Abtastung im Zeitbereich entspricht einer Multiplikation mit einer Dirac-Stoßfolge [8, S. 153]

$$x_A(t) = x(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - nT) = x(t) \cdot \delta(t - nT) = x[n] \tag{2.21}$$

Die technische Realisierung der Abtastung kann mit einem sogenannten Sample and Hold-Glied (SH) durchgeführt werden. Durch die Abtastung wird das analoge Eingangsignal im Zeitbereich diskret. Die Abtastung hat im Frequenzbereich eine periodische Fortsetzung des Eingangsspektrum periodisch zur Folge. Es werden bei ganzzahligen vielfachen der Abtastfrequenz f_s jeweils ein Spiegelspektrum und das Eingangsspektrum abgebildet. Die zeitliche Abtastung eines Signal kann unter bestimmten ein reversibler Prozess sein. Es ist mit einem entsprechend dimensionierten D/A-Wandler möglich das Eingangssignal ohne Fehler im Zeitbereich wiederherzustellen. Eine Bedingung dafür wird als Nyquist-Abtasttheorem bezeichnet.

$$f_{max} \leq \frac{f_s}{2} \tag{2.22}$$

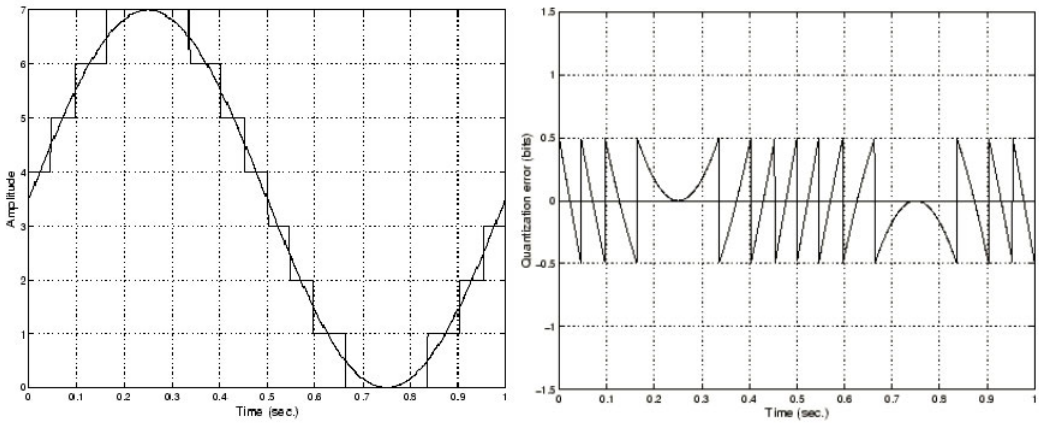
Nach (2.22) muss für eine fehlerfreie Rekonstruktion im Zeitbereich die Abtastrate des SH-Gliedes mindestens doppelt so groß sein, wie die höchste im Signal vorhandene Frequenz sein. So wird die spektrale Überlappung (Aliasing) des abgetasteten Signals verhindert (siehe Abbildung **??**).
Je größer die Abtastrate ist, desto weiter sind die periodischen Spektren voneinander entfernt. Um

die maximal enthaltene Signalfrequenz zu begrenzen und somit Aliasing-Artefakte zu verhindern wird das Eingangssignal vor der A/D-Wandlung mit einem Anti-Aliasing-Filter (Tiefpass-Filter) gefiltert.
Zusätzlich zum reversiblen Prozess der zeitlichen Abtastung muss das analoge Signal für die digitale Signalverarbeitung auch noch im Amplitudenwertebereich diskretisiert werden. Dieser Vorgang wird als Quantisierung bezeichnet. Der Fehler der beim Quantisieren entsteht ist nicht reversibel und äußert sich als zusätzliches Rauschen. Das Quantisierungsrauschen entsteht durch Rundungsfehler aufgrund der Begrenzung des unendlichen Wertebereichs in einen diskreten Wertebereich mit einer endlichen Anzahl von Amplitudenwerten. Die Leistung des Quantisierungsrauschen e hängt dabei von der Anzahl der Quantisierungsstufen bzw. der kleinsten darstellbaren Amplitude (q,Auflösung) ab.

$$P_{Quantisierung} = \frac{1}{q} \cdot \int_{-\frac{q}{2}}^{\frac{q}{2}} e^2 de = \frac{q^2}{12} \tag{2.23}$$

Das Quantisierungsrauschen wird durch jeden A/D-Wandler hinzugefügt und kann nur verringert, jedoch nicht verhindert werden (siehe Abbildung 2.10). Wenn das Eingangssignal den gesamten gül-

Abbildung 2.10: Vergleich des Signalverlaufs eines zeitdiskreten Signals mit hoher und niedriger Amplitudenauflösung (links) und des dazugehörigen Quantisierungsfehlers (rechts)[7]



tigen Eingangsbereich des A/D-Wandler abdeckt ist der A/D-Wandler voll ausgesteuert. Der Quantisierungsfehler ist unter dieser Bedingung nahezu gleichverteilt mit Werten von $\pm q/2$ (siehe Abbildung 2.10) und besitzt einen Mittelwert gleich Null. Sollte das Signal den A/D-Wandler übersteuern clippt das quantisierte Signal. Am Eingang des A/D-Wandler liegt in diesem Fall eine zu hohe Amplitude vor, welche nicht mehr linear mit einem gültigen Codewort abgebildet werden kann. Die hohen Amplituden werden auf den höchsten darstellbaren Wert abgebildet. Für sehr hohe Amplituden, die außerhalb des Wertebereichs des A/D-Wandler liegen entsteht ein rechteck-ähnlichen Ausgangssignal mit nicht-linearen Frequenzkomponenten.
Um das Quantisierungsrauschen zu verringern kann die Noiseshaping-Technik angewendet werden. Sie wurde bereits in Zusammenhang mit dem $\Delta\Sigma$-Wandler erwähnt und beschreibt die spektrale Formung des Quantisierungsrauschens. Aus der Kombination des Noiseshapings und der Überabtastung wird das effektiven SNR der Analog-Digital-Wandlung durch einen $\Delta\Sigma$-Wandler wesentlich verbessert.
Die A/D-Wandlung liefert aus einem analogen Eingangsignals ein digitales Ausgangssignal mit einem diskreten Amplituden- und Zeitbereich. Diese Art der Signale können im Computer gespeichert, weiterverarbeitet und analysiert werden.
Ein Analyse-Möglichkeit von digitalen Signalen ist die FFT-Analyse. Die FFT stellt ist ein optimierte Berechnungsalgorithmus für die DFT. Mit der diskreten Fouriertransformation ist es möglich das

Spektrum eines zeitdiskreten Signales zu berechnen. Die vollständige Fouriertransformation lautet

$$X_A(j\omega) = \sum_{n=-\infty}^{\infty} x[n] \cdot \mathrm{e}^{-jn\omega T} \ . \tag{2.24}$$

Um das Spektrum des Signals zu berechnen wäre eine vollständige Betrachtung von $-\infty$ bis ∞ notwendig. In der Realität lässt sich das nicht umsetzten und so wird der Bereich des Signals, welcher zu Analyse genutzt wird, auf maximal N Werte eingeschränkt. Die Berechnungsvorschrift der DFT lautet (siehe Gleichung (2.25))):

$$X[m] = \sum_{n=0}^{N-1} x[n] \cdot \mathrm{e}^{-j2\pi \frac{mn}{N}} \tag{2.25}$$

Die inverse Rechenvorschrift zur Berechnung des Zeitsignals $x[n]$ aus dem Spektrum $X[m]$ wird als IDFT bezeichnet und in Gleichung (2.26) bezeichnet.

$$x[n] = \frac{1}{N} \sum_{m=0}^{N-1} X[m] \cdot \mathrm{e}^{j2\pi \frac{mn}{N}} \tag{2.26}$$

Das DFT-Spektrum ist diskret und periodisch. Die Berechnung der DFT bei der Frequenz m benötigt genau N komplexe Multiplikationen. Das gesamte DFT-Spektrum erfordert N^2 Multiplikationen. Die Berechnung eines Spektrums kann deshalb eine hohe Berechnungszeit benötigen. Um das Ergebnis in angemessener Zeit zu berechnen wurden verschiedenen Algorithmen entwickelt um die Rechenzeit durch Reduzierung der Anzahl der Multiplikationen zu verkürzen. Aufgrund der Periodizität der harmonischen Funktionen sind viele Zwischenergebnisse identisch. Der FFT-Radix 2 Algorithmus vermeidet die Berechnung der redundanten Terme der DFT und benötigt nur N komplexe Multiplikationen [8, S. 175].
Die Analyse eines Zeitsignals mit der FFT fügt aufgrund der Betrachtung eines endliche langen Zeitraumes (Zeitfensterung durch Betrachtung von maximal N Samples) immer einen gewissen Fehler hinzu. Die FFT bezieht sich auf ein periodisch fortgesetztes Signal. Besitzen Signalanfang und -ende einen hohen Unterschied in ihrer Amplitude entstehen hochfrequente Signalanteile aufgrund der periodischen Fortsetzung des Zeitfenster. Diese Signalanteile sind somit künstlich durch die Fensterung dem Signal hinzugefügt worden. Dieser Effekt wird auch als spektraler Leck-Effekt bezeichnet. Durch die Gewichtung des Signale innerhalb eines zeitlich begrenzten Analysefensters kann dieser effektiv verringert werden.
Es gibt viele verschiedene Fensterfunktionen die verwendet werden können. Sie unterscheiden sich im maximal möglichen Amplitudenfehler, der Hauptkeulenbreite, und der Höhe der Nebenkeulen. Welches Fenster für eine Anwendung verwendet wird hängt von jeweiligen Aufgabenstellung ab. Zwei wichtige Fensterfunktionen sind das Hanning-Fenster und das Flat-Top-Fenster (siehe Gleichungen (2.27) und (2.28) nach [8]).

$$w[n] = 0,5 - 0,5 \cdot \cos\left(\frac{2\pi n}{N}\right) \tag{2.27}$$

$$w[n] = a_0 - a_1 \cdot \cos\left(\frac{2\pi n}{N-1}\right) + a_2 \cdot \cos\left(\frac{4\pi n}{N-1}\right) - a_3 \cdot \cos\left(\frac{6\pi n}{N-1}\right) + a_4 \cdot \cos\left(\frac{8\pi n}{N-1}\right) \tag{2.28}$$

Mit $a_0 = 0,2155, a_1 = 0,4159, a_2 = 0,2780, a_3 = 0,0836, a_4 = 0,0070$. Das Flat-Top-Fenster kann immer verwendet werden, wenn eine hohe Amplitudengenauigkeit bzw. geringer Amplitudenfehler notwendig ist [31]. Der Amplitudenfehler des Flat-Top-Fensters liegt praktisch bei Null wo hingegen das Hanning-Fenster einen Amplitudenfehler von etwa $-1,5\ dB$ besitzt. [8, S. 195]. Demgegenüber steht die frequenzselektive Auflösung der jeweiligen Fenster. Nach [8] weist das Flat-Top-Fenster

die schlechteste Frequenzauflösung auf. Einen angemessenen Kompromiss zwischen maximalen Amplitudenfehler und Frequenzauflösung stellt das Hanning-Fenster dar. Gerade bei der Analyse unbekannter Signale empfiehlt es sich dieses Fenster zu benutzen.
Die FFT-basierte Analyse umfasst nach Ansicht von Meyer [8] sechs Bearbeitungsschritte (siehe Abbildung 2.11).

Abbildung 2.11: Darstellung der prinzipiellen Elemente eines FFT-Prozessors[8]

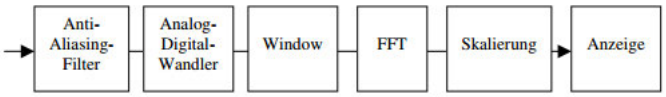


Bild 5.33 Blockschema des FFT-Prozessors

Die ersten beiden Schritte (Anti-Aliasing-Filter und A/D-Wandler) wurden bereits mit der Hardware-Beschreibung besprochen. Die Notwendigkeit der Fensterung wurde in diesem Abschnitt erläutert und auch zwei verschiedene Fenster vorgestellt. Genauso wie die Berechnung der FFT. Das Ergebnis der Berechnung des Spektrum liefert ein Zwei-Seiten-Spektrum mit N-Punkten. Das Ergebnis ist komplex und enthält Informationen sowohl über die Amplituden, als auch der Phase der Signals. Die Frequenzauflösung (Abstand der Spektrallinien zu einander) von $\Delta f = \frac{f_s}{N}$. Die Signalinformationen liegen im Frequenzbereich von $0Hz$ bis $\frac{f_s}{2}$.

2.4. ARRAYMESSTECHNIK

Zur Untersuchung von Schallfeldern ohne Rückwirkung auf die Quelle können Mikrofonarray-Systeme (wird auch als akustische Antenne bezeichnet) verwendet werden. Sie können je nach Dimensionierung Schallquellen lokalisieren und die Stärke des Quelle bestimmen.
Das Grundprinzip beruht auf der Auswertung der Laufzeit- und Pegelunterschiede zwischen den verteilten Mikrofonen. Dieses Prinzip wird auch Beamforming, die räumliche diskrete Abtastung durch mehrere Sensoren, genannt. Um eine möglichst hohe räumliche Abstastung zu bekommen müssen dem entsprechend viele Mikrofone verwendet werden. Je nach Messort und Messumgebung kann die Anzahl der Sensoren variieren.
Bei der Arraymesstechnik ist neben dem zeitlichen Abtasttheorem auch ein räumliches Abstasttheorem zu beachten

$$d \leq \frac{\lambda}{2} \text{ für } \Theta = 90° \tag{2.29}$$

Die Wellenlänge muss nach (2.29) doppelt so groß sein wie der Abstand der Mikrofone zu einander. Daraus ergibt sich auch der maximale mögliche Aufnahmewinkel für das Array.
Die Fokussierung des Arrays kann elektronisch vorgenommen werden. Eine Variante der elektronischen Fokussierung ist im Delay-and-Sum-Algorithmus implementiert. Dabei wird für ein Array und einen bestimmten Fokussierungswinkel die theoretischen Laufzeitverzögerungen für jedes Mikrofon berechnet [2]. Die so ermittelte Zeit werden anschließend für das jeweilige Mikrofonsignal kompensiert und diese summiert. Dieser Vorgang wird für alle weiteren Fokussierungswinkel wiederholt. In Senderichtung kommt es zu konstruktiven Überlagerungen der Messsignale. Die Pegel sind somit in der Richtung der Schallquelle am größten.
Die Richtwirkung eines Mikrofonarrays ist im wesentlichen abhängig von dem Verfahren und nicht von den Sensoren oder der nachfolgenden DAQ-Hardware. Die Antwortfunktion eines Array ist eine wichtigere Größe wenn mehr als eine Quelle gleichzeitig vorhanden ist, weil das Ergebnis einer Array-Messung aus der Summe der Faltungen der Array-Antwortfunktion mit jeder vorhandenen Quelle entsteht. Besitzt die Antwortfunktion eine hohe Nebenkeulenhöhe kann eine leise Schallquelle von eine in der Nähe befindlichen lauten Schallquelle überdeckt werden.

Um das Ergebnis des Beamforming zu verbessern ist es möglich die Haupt- und Nebenkeulenhöhen und -breiten zu verändern, indem die Signale der einzelnen Mikrofone gewichtet werden [32]. Von praktischen Interesse ist weiterhin interessant, dass die Mikrofone nicht nur in der Amplitude un Phase kalibriert sein müssen. Es gilt auch eine zulässigen Positionsfehler nicht zu überschreiten, was insbesondere in unwegsamen Gelände schwierig zu realisieren sein kann. Der zulässige Positionsfehler beträgt

$$\Delta x = \frac{\lambda}{36} \tag{2.30}$$

um den dazugehörigen Phasenfehler auf maximal 10° zu begrenzen [32].

3

Entwicklung eines mobilen Datenerfassungssystems

Für eine Forschungsexpedition im Mai 2014 ist eine akustische Messung der Stromboli Vulkan auf den liparischen Inseln in Italien mit einem 16-kanaligen Mikrofonarray geplant gewesen. Um die Signale aufzunehmen sollte im Rahmen dieser Arbeit ein Datenerfassungssystem entwickelt werden welches die Signale von insgesamt 16 Mikrofonen erfasst und speichert.
In diesem Kapitel wird zunächst der zeitliche Ablauf der Arbeit dargestellt. In Gesprächen mit den zuständigen Verantwortlichen Herr Prof. Dr. Sesterhenn und Frau Stampka M.Sc. wurden die verschiedenen Anforderungen an das Datenerfassungssystem besprochen und regelmäßig reflektiert. Die endgültigen Gesamtanforderungen werden anschließend im Abschnitt 3.1 vorgestellt.
Diese Arbeit fokussierte sich besonders auf die Zusammenstellung geeigneter DAQ-Hardware und Entwicklung einer passenden DAQ-Software-Applikation. Die Auswahl der Mikrofone, sowie der Array-Geometrie und das Beamforming sind nicht Gegenstand dieser Arbeit gewesen.

3.1. Anforderungen und zeitlicher Ablauf der Entwicklung

Die Arbeit an diesem Projekt begann Anfang Januar 2014. Es fanden erste Gespräche mit den Projektverantwortlichen statt und die Erfahrungen der letzten Forschungsreise wurden ausgewertet. Daraus leiteten sich diverse Anforderungen das Datenerfassungssystems ab. Für die Entwicklung eines passenden Datenerfassungssystems musste zuerst nach geeigneten Komponenten recherchiert werden.
Zur Beschaffung der DAQ-Komponenten stand ein Budget von insgesamt 20.000 € abzüglich der Kosten für zusätzliche akustische Sensoren und Messgeräte für Umgebungsdaten zur Verfügung. Bei den Mikrofonen sollte das bestehende Inventar von zehn PCB 378B02 Mikrofonen auf insgesamt 17 Mikrofone aufgestockt werden. Die 7 benötigten PCB 378B02 Mikrofone kosteten 6.300 €, und ließen ein effektives Budget von 13.700 £ für ein 16-kanaliges Datenerfassungssystem und Zubehör übrig lassen.
In einer ersten Planungsstufe wurde ein X-Array mit ein Größe von 20 m x 20 m geplant. Der Messort befindet sich ca. 900 m über dem Meeresspiegel. Es gibt keine direkte Zufahrt zum Vulkan. Der Aufstieg erfolgt über einen ca. dreistündigen Wanderweg. Auf dem Gipfel angekommen soll das Datenerfassungssystem einen zehnstündigen Dauerbetrieb ermöglichen. Das ist nur mit eine ausreichenden Akkuleistung und einem geringen Stromverbrauch möglich.
Bis Anfang Februar wurde nach passenden Systemkomponenten recherchiert. Neben den finanziellen Anforderung mussten natürlich elektrische und mechanische Eigenschaften berücksichtigt werden.
Das DAQ-System wird hauptsächlich auf dem Rücken und mit dem Flugzeug im Handgepäck trans-

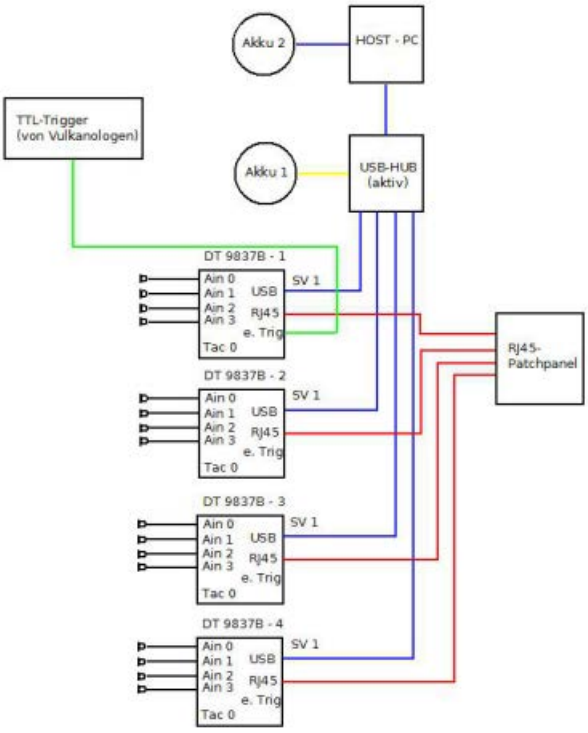
portiert. Aus diesem Grund darf das Datenerfassungssystem ohne Kabel maximal 8 kg wiegen und die Abmessungen von 55 cm x 40 cm x 23 cm x nicht überschritten werden. Die Kabel werden im Frachtraum transportiert und vor Ort von einer zweiten Person getragen.
Aus den Erfahrungen von der letzten Expedition [12] war bekannt das der interessierende Frequenzbereich für die Messung im Bereich von 10 Hz bis 2 kHz lag. Das Datenerfassungssystems sollte jedoch nicht nur für die Vulkan-Messung zum Einsatz kommen, sondern auch für weitere Projekte des Institut für Numerische Fluiddynamik der TU Berlin genutzt werden können. Deshalb muss die DAQ-Hardware auch Eingangssignale im Frequenzbereich bis zu 50 kHz verarbeiten können. Die untere Grenzfrequenz soll laut Aussage von Herrn Prof. Dr. Sesterhenn bei höchstens $0,1$ Hz liegen. Damit sollte sichergestellt werden, das auch Infraschall-Sensoren an das System angeschlossen werden können.
Der Frequenz- und Phasengang der Hardware muss in diesem Bereich möglichst linear verlaufen. Das es sich um ein mehrkanaliges System handelt muss durch die Hardware auch eine synchrone und sample-genaue Verarbeitung der 16 Eingangskanälen sicherstellen. Für jeden Eingangskanal ist eine IEPE kompatible Signalkonditionierung notwendig, da die bereits vorhanden Mikrofone extern vorpolarisiert sind und nur mit einem Konstantstrom funktionieren. Die maximale Ausgangsspannung der PCB 378B02 Mikrofone beträgt $U_{max} = \pm 7\ V$. Der A/D-Wandler muss mindestens einen genauso großen Eingangsbereich besitzen um einer Übersteuerung vorzubeugen und nicht lineare Verzerrungen zu vermeiden. Damit verbunden ist auch die Forderungen nach einem gutem SNR und geringen Klirrfaktor (THD).
Für die Recherche und Entwicklung der Datenerfassung standen insgesamt 5 Monate zur Verfügung, da die Expedition im Mai 2014 beginnen sollte. Bis dahin musste das gesamte Datenerfassungssystem inklusive Software fertig gestellt sein. Aufgrund der unbekannten Lieferzeiten und des relativ geringen Zeitraumes zur Entwicklung und Evaluierung wurde die Idee eines Eigenbau des Datenerfassungssystems verworfen. Durch die relativ hohen Anzahl der benötigten Kanäle (und des zur Verfügung stehenden Budget) war es auch nicht möglich ein Komplett-System zu benutzen. Die Recherche der Komplett-Systeme zeigte, das teilweise nicht alle Anforderungen durch die Komplett-Systeme erfüllt wurden. Gängige Systeme von Herstellern sind z.B die LAN XI Hardware von Brüel & Kjær, das HEAD*lab* von HEADacoustics, das NI-CompactDAQ-Chassis von National Instruments, das OAK MKII von Müller-BBM oder das Cronosflex von imc.
Die Recherche zeigte, das ein wesentlicher Nachteil der Komplettsystem in ihrem hohen Gewicht, einem relative großen Energieverbrauch und einem eingeschränkten Frequenzbereich besteht. Im Zuge der Recherche wurde der Einsatz modularer USB-Messboxen aufgrund des geringeren Gewichts und Strombedarf bevorzugt recherchiert. Es gibt zahlreiche Anbieter von gleichwertigen USB-Messboxen z.B. SINUS Messtechnik GmbH, ROGA Instrument, disynet GmbH oder die GOLDAMMER GmbH. Unter Berücksichtigung der Anforderungen zeigte das Angebot vier-kanaliger USB-Messgeräte der Firma Data Translation angemessene Möglichkeit zum Einsatz im DAQ-System. Andere Systeme besaßen nur eine Auslösung von 16-Bit oder eine zu geringe Abtastrate bzw. eine zu hohe untere Grenzfrequenz.
Um festzustellen ob die USB-Messgeräte von Data Translation wirklich geeignet waren wurde eine Teststellung gemacht um die verschiedenen Parameter und die Benutzeroberfläche der DT9837 Module in einem ersten Test überprüfen zu können. Ein zweiter Schwerpunkt lag in der Untersuchung wie sich die DT9837 Module mit MATLAB programmieren lassen. Die Teststellung traf Anfang Februar ein und stand zwei Wochen zu Untersuchung zu Verfügung.
Anfang März lagen die Ergebnisse der Teststellung vor und es konnten verschiedene Konfigurationen des Datenerfassungssystems besprochen werden. Es wurden unter anderem die verschiedenen Vor- und Nachteile unterschiedlichen Messsysteme besprochen. Außerdem wurde der Einsatz eines Einplatinen-Rechner als Host-PC geklärt. Bei diesem Gespräch wurde auch entschieden das bei dem 16-kanaligen System ein Mikrofon ein Infraschall-Mikrofon sein sollte. Das Infraschall-

Mikrofon soll vergleichbare Ergebnisse zu anderen akustischen Vulkan-Studien ermöglichen, die im wesentlichen eine Analyse im Infraschall-Bereich ($f \leq 20\ Hz$) vorgenommen haben.
Mitte März 2014 wurde die Bestellung von vier DT9837B Modulen in Auftrag gegeben. Auf Anfrage wurde eine Sonderfertigung mit einem $0,1\ Hz$ Hochpass-Filter geliefert. Das Datenerfassungssystem arbeitet PC-gestützt und kann mit der Data Acquisition Toolbox von MATLAB gesteuert werden. Eine Software, welche die Messaufgaben für die Expedition erfüllt, gab es nicht. Diese wurde für die Expedition erstellt und an das System angepasst. Die Entwicklung der Messsoftware erfolgte von Anfang April bis Anfang Mai. Es wurde eine komplette Nutzeroberfläche erstellt, da die mitgelieferte Software nicht die gewünschten Funktionen zur Verfügung stellte und nicht flexibel anpassbar war.

3.2. MOBILE ARRAY MEASUREMENT SYSTEM

Das PC-gestützte Datenerfassungssystems MAMS misst die elektrischen Signale von bis zu 16 Eingangskanälen. Die dazugehörige DAQ-Software ist als Programm in MATLAB, unter Verwendung der Data Acquisition Toolbox, implementiert. Der Projektname des DAQ-Systems lautet *Mobile Array Mesurement System*. Die Abbildung 3.1 zeigt die Verbindungen der einzelnen Komponenten des MAMS untereinander.

Abbildung 3.1: Übersicht über das MAMS



Das MAMS besteht aus einer Kombination aus USB-Messgeräten und einem Host-PC mit einer MATLAB-basierten Messsoftware. Die verwendeten Mikrofone werden an die Eingangsbuchsen der USB-Messgeräte (DT9837B) angeschlossen. Diese verfügen über eine entsprechende Signalkonditionierung für die Mikrofone. In den DT9837B-Modulen sind vier parallel arbeitende A/D-Wandler integriert. Sie wandeln die elektrischen Signale in digitale um. Die Verbindung der Module mit den RJ45-Panel soll sicherstellen, dass alle Module synchron arbeiten. Die Module sind über einen USB-Hub an den Host-PC mit der Messsoftware angeschlossen. Die Stromversorgung wird durch die in-

ternen Akkus des USB-Hubs und der Akkus des Host-PC sichergestellt. Außerdem steht eine externe Powerbank zur Verfügung.
Bis kurz vor Beginn der Expedition war die Verbindung einer TTL-Quelle von den italienischen Vulkanologen mit dem MAMS vorgesehen. Diese Entscheidung wurde kurzfristig vom Projektverantwortlichen revidiert und die Funktion deshalb aus der Software entfernt.
Durch die Verwendung eines mobilen Host-PC mit einem Windows-Betriebssystems ist es auch möglich die Messung mit einem Remote-PC per LAN-Kabel oder WLAN-Verbindung zum Host-PC fernzusteuern. Das ist besonders bei unzugänglichen oder weit entfernten Messorten vorteilhaft.

3.2.1. SENSOREN

Für die Messung der akustischen Signale der Geräusche am Vulkan kamen moderne vorpolarisierte Kondensatormikrofone zum Einsatz. Bereits während der Expedition in 2013 [12] wurden diese Mikrofone eingesetzt. Für eine gute Vergleichbarkeit wurde weitere Einsatz der ICP-Mikrofone PCB 378B02 von PCB Piezotronics auch für die Expedition in 2014 beschlossen. Am ISTA waren bereits zehn Exemplare dieser Mikrofone vorhanden. Für die Array-Messung wurden zusätzlich sieben weitere baugleiche Mikrofone bestellt. Insgesamt standen für die Messung am Vulkan 17 PCB 378B02 zur Verfügung.
Die akustische Messung mit einem Mikrofonarray von Vulkanen in Nahbereich mit einer Entfernung von weniger als 500 m besitzt kaum vergleichbare Messergebnisse. Die meisten akustischen Messungen untersuchen die Geräusche von Vulkanen mit einer geringen Abtastrate im Bereich unterhalb von 20 Hz. Um die Ergebnisse der Messung am Stromboli im Mai 2014 mit anderen Forschungsergebnisse vergleichen zu können wurde ein zusätzlicher Infraschall-Sensor für die DAQ-Hardware vorgesehen werden. Nach einer ausführlichen Recherche wurde eine Kombination aus Mikrofonkapsel und Mikrofon-Vorverstärker der Firma G.R.A.S ausgewählt. Die vorpolarisierte Mikrofonkapsel 40AZ besitzt eine untere Grenzfrequenz von $0,2\ Hz$ in Kombination mit dem CCP-Vorverstärker 26CA.
Für die nachfolgende DAQ-Hardware ist es wichtig genau zu wissen elektrischen Signale am Ausgang des Mikrofonvorverstärkers zu erwarten sind. Eine Zusammenfassung der wichtigsten zu berücksichtigen Eigenschaften der Mikrofone sind in der Tabelle 3.1 dargestellt.

Tabelle 3.1: Zusammenfassung der wichtigsten elektrischen und mechanischen Merkmale der PCB 378B02 Mikrofone

	PCB 378B02	G.R.A.S 40 AZ + G.R.A.S. 26CA
Durchmesser der Mikrofonmembran	$1/2 Zoll$	$1/2 Zoll$
Mikrofonentzerrung	Freifeld	Freifeld
nomineller Übertragungsfaktor	$50\ mV/Pa$	$50\ mV/Pa$
Frequenzbereich ($\pm 1\ dB$)	$7 - 10\ kHz$	$1 - 10\ kHz$
Frequenzbereich ($\pm 2\ dB$)	$3.75 - 20\ kHz$	$0,5 - 20\ kHz$
untere Grenzfrequenz ($-3\ dB$)	$1 - 3\ Hz$	$0,2\ Hz$
Dynamikbereich	$\geq 135\ dB_{re\ 20\ \mu Pa}$	$\geq 134\ dB_{re\ 20\ \mu Pa}$
Betriebstemperaturbereich	$-40 - 80°C$	$-40 - 120°C$
Polarisationsspannung	$0\ V$ (vorpolarisiert)	$0\ V$ (vorpolarisiert)
Kapazität der Mikrofonkapsel	$12\ pF$	$20\ pF$
Signalkonditionierung	ICP (IEPE)	CCP (IEPE)
Arbeitspunktspannung	$10 - 14\ VDC$	-
Ausgangsimpedanz	$\leq 50\ \Omega$	$\leq 50\ \Omega$
maximale Ausgangsspannung	$\pm 7\ V_{pk}$	$\pm 8\ V_{pk}$
elektrischer Verbinder	BNC-Steckverbinder	BNC-Steckverbinder

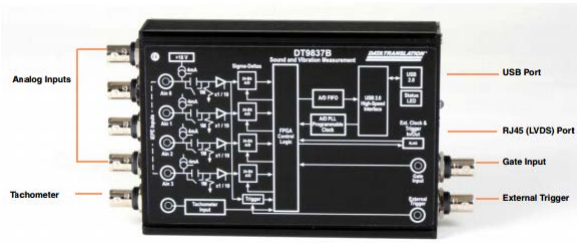
Die vorpolarisierten Mikrofone stellen den aktuellen Stand der Technik dar. Sie sind aufgrund

der Konstantstromtechnik unempfindlicher gegenüber Änderungen der Kabellänge und- Kapazität. Durch die geringen Ausgangsimpedanzen sind die Verluste auf langen Messstrecken ebenfalls gering. Mit diesen Eigenschaften ist es möglich kostengünstige Messkabel z.B. Koaxialkabel für die Messung verwendet werden. Außerdem sind die Kosten für die benötigte Signalkonditionierung gering, weshalb die Kosten pro Kanal insgesamt relativ gering ausfallen.
Das PCB 378B02 ICP-Mikrofon weist einen breiten Übertragungsbereich ($-3\ dB$) von etwa 3 Hz bis 20 kHz auf. Um in tieffrequenten Bereich unterhalb von 3 Hz messen zu können wird das Mikrofon von G.R.A.S mit einer unteren Grenzfrequenz von $0,5\ Hz$ verwendeten. Be beiden Mikrofonen handelt es sich um IEPE-Mikrofone mit der jeweiligen Herstellerbezeichnungen ICP und CCP. Ein wesentlicher Unterschied beider Mikrofone die maximale Ausgangsspannung der Sensoren. Die ICP-Mikrofone liefern maximal eine Ausgangssignals $\pm 7\ V$. Das CCP-Infraschallmikrofon liefert ein 1 Volt höheres Ausgangssignal mit Werten von $\pm 8\ V$.

3.2.2. DATENERFASSUNG

Die Recherche nach passender DAQ-Hardware zeigte, das modulare USB-Messboxen am geeignetsten für die bevorstehende Messkampagne sind. Es gibt Module mit einer hohen Abtastrate und einem großen linearen Frequenzbereich. Einige Module zeigen ein zufriedenstellendes Rauschverhalten, wobei einige Module auch ein relativ starkes Eigenrauschen vorweisen. Nach der Teststellung im Februar 2014 die wurde die Entscheidung für die USB-Messgeräte der Firma Data Translation getroffen. Durch die flexible Programmierbarkeit mittels MATLAB-Programmcode und technischen Angaben des Datenblatts stellten diese Messmodule eine gute Wahl dar. Die USB-Module sind relativ leicht und besitzen sehr kompakte Abmessungen. Das schwarze Gehäuse besitzt insgesamt 7 BNC-Steckbuchsen, einen RJ45 Port und einen USB 2.0 Port (siehe Abbildung 3.2).

Abbildung 3.2: Blockdiagramm der wesentlichen Komponenten der DT9837B-Module



Auf den Geräten ist jeweils auch das Blockschaltbild des DAQ-Moduls aufgedruckt. Bei den hier vorgestellten Datenerfassungssystem werden insgesamt vier USB-Messmodule eingesetzt. Es handelt sich dabei um eine spezielle Variante der DT9837B-Module[1]. Das DT9837B-Modul besitzt vier analoge single-ended BNC-Eingänge an die Sensoren angeschlossen werden können. Ein zusätzlicher Tachometer-Eingang steht ebenfalls zur Verfügung. Dieser findet für das MAMS keine Verwendung. Die Abbildung 3.2 zeigt, dass jede Eingangsstufe der vier BNC-Eingänge aus einem BNC-Steckverbinder, einer IEPE-kompatiblen Konstantstromquelle, einem zuschaltbaren Koppel-Kondensator, einem Operationsverstärker und einem $\Delta\Sigma$-Wandler besteht. Eine FPGA-Steuerlogik verarbeitet die gewandelten Daten und leitet sie an die USB-Buchse des Moduls, zur Analyse am Host-PC, weiter. Um eine hohe Genauigkeit zu erhalten werden die digitalen Signale der A/D-Wandler in einem FIFO-Buffer zwischen gespeichert. Dadurch können keine Samples verloren gehen. Eine Phasenregelschleife (PLL) leitet aus dem USB-Takt die interne Taktrate des Moduls ab. Die interne Taktrate der Module ist gleichzeitig die Frequenz mit der die $\Delta\Sigma$-Wandler die elektrischen Ein-

[1] Auf speziellen Wunsch hat Data Translation den Hochpassfilter dr DT 9837B Variante gegen einen Filter mit einer Grenzfrequenz von $0,1\ Hz$ ausgetauscht. Damit stand eine Kombination aus maximale Abtastrate pro Kanal von $105,4\ kHz$ und unterer Grenzfrequenz von $0,1\ Hz$ zur Verfügung.

gangssignale abtasten. Wie bereits im Grundlagen-Kapitel erklärt erreichen die ΔΣ-Wandler ihre Performance unter anderem durch einen hohen Oversampling-Faktor. Die DT9837B-Module können einen maximalen Takt von 27 MHz erzeugen.

SIGNALKONDITIONIERUNG UND A/D-WANDLUNG

Es ist möglich für jede Eingangsbuchse eine zuschaltbare IEPE-Konditionierung einzuschalten. Die Versorgungsspannung der Konstantstromquelle der DT9837B-Module beträgt 18 V mit einem Strom von 4 mA. Die analogen Eingangssignale werden mit einem 24-Bit ΔΣ-Wandler in digitale Signale umgewandelt. Die Module wurden entsprechend der Anforderung durch die Mikrofone ausgesucht. Das Infraschall-Mikrofon besitzt einen größeren Ausgangsbereich als die ICP-Mikrofone. Die maximale Ausgangsspannung liegt zwischen $-8\ V$ bis $+8\ V$. Die Arbeitspunktspannung beträgt laut Herstellerangaben ca. 10 V. Die untere Aussteuergrenze sollte 1 V nicht unterschreiten um negatives clippen zu verhindern. Der untere Aussteuerungsbereich umfasst

$$NegativeRange = V_{Bias} - \ V = 10\ V - 1\ V = 9\ V \tag{3.1}$$

Der obere Aussteuerungsbereich wird durch die Versorgungsspannung der Kosntantstromquelle bestimmt.

$$positiveRange = (V_{source} - 1) - V_{Bias} = 18\ V - 1\ V - 10V = 7\ V \tag{3.2}$$

Es zeigt sich, dass der obere Aussteuerungsbereich des A/D-Wandlers um 1 V kleiner als der maximale Ausgangsbereich des Infraschallmikrofon von G.R.A.S ist. Mit einer typischen Mikrofonempfindlichkeit von 50 mV/Pa erfolgt ein Clippen des Infraschall-Mikrofons ab einem Schalldruck von $140dB$. Nach [33] ist die Schallausbreitung in Luft bis zu einem Pegel von 130 dB als linear zu betrachten. Oberhalb dieses Pegeln treten zunehmend nichtlineare Effekte auf. Aus diesem Grund wurde der um 1 V verringerte Aussteuerungsbereich als akzeptabel eingestuft. Außerdem sind Pegel oberhalb von 130 dB aufgrund der Erfahrungen der vergangenen Expedition [12] nicht zu erwarten. Um das Messsignal von der Gleichspannung im Bereich der Arbeitspunktspannung zu trennen wird das Signal mit einem Hochpassfilter gefiltert. Der Gleichspannungsanteil wird durch das Filter ausgesiebt und stellt dem A/D-Wandler anschließend nur das Wechselspannungssignal im Bereich von $-7\ V$ bis 8 V zur Verfügung.
Die Analog-Digital-Wandlung erfolgt durch einen überabtastenden ΔΣ-Wandler. Der Eingangsbereich der Wandler kann für jeden Kanal softwareseitig auf zwischen $\pm 10\ V$ und $\pm 1\ V$ umgeschaltet werden. Die maximale Abtastfrequenz beträgt für diese Gerätevariante $f = 105,469\ kHz$. In Abhängigkeit der Abtastrate wird die Grenzfrequenz des Anti-Aliasing-Filter gesetzt. Die Grenzfrequenz des Filters liegt bei $0,49 \cdot f_s$. Der gesamte Übertragungsbereich des DT9837B verläuft somit von $0,1\ Hz$[2] bis $0,49\ cdot f_s$. Für eine Abtastrate von 48 kHz bedeutet dies, dass Frequenzen von $0,1\ Hz$ bis $23,520\ kHz$ ausgewertet werden können.
In Abhängigkeit der Abtastrate wird neben der Grenzfrequenz des Antialiasing-Filters auch die interne Clock-Rate gesetzt. Wenn die vom Benutzer gewünschte effektive Abtastrate kleiner als $52,7\ kHz$ ist, beträgt der Oversampling-Faktor $r = 512$. Liegt die Abtastrate höher als $52,7\ kHz$ beträgt der Oversampling-Faktor $r = 256$. Bei einem Messsystem mit $f_s = 48\ kHz$ beträgt der SNR allein durch die Überabtastung bereits $81,27dB$. Bedingt durch das Wandlerprinzip ist der ΔΣ-Wandler mit einer erhöhten Gruppenlaufzeit verbunden. Diese beträgt für die DT9837B-Module insgesamt 39 Samples. Diese Laufzeitverzögerung ist für alle Kanäle gleich und wird durch die Geräte-Software kompensiert.

[2] Wenn der Koppelkondensator zu geschaltet ist.

STROMBEDARF

Die Stromversorgung der Messmodule wird über die USB-Buchsen des Host-PCs sichergestellt. Die DT9837B Variante verfügt über einen USB 2.0 Anschluss mit Hot Swap-Funktion[3]. Gemäß den Spezifikationen [LITERATURVERWEIS] wird pro USB 2.0 Port eine maximaler Strom von 500 mA bei einer Gleichspannung von $\pm$ 5 V. Die maximale zur Verfügung stehende Leistung an einem USB 2.0 Port beträgt demnach 2,5 W. Laut Herstellerangaben [9] beträgt die typische Leistungsaufnahme der Module 0,425 A bei $\pm$0,3 V was einer Leistung von 0,1275 W entspricht (siehe (3.3)).

$$P_{DT9837B} = 0,425\ A \cdot 0,3\ V = 0,1275\ W \tag{3.3}$$

MEHRKANAL-MESSUNG

Ein DT9837B USB-Messgerät besitzt vier Eingangskanäle zur Messwerterfassung. Um darüber hinaus weitere Sensoren anzuschließen besteht die Möglichkeit bis zu drei weitere DT9837B-Module an das vorhandene Messgerät anzuschließen. Dazu werden die Module über den Synchronisations-Eingang am Modul elektrisch miteinander verbunden. Die Sync-Buchse ist als RJ45(LVDS)-Port ausgeführt. Um alle Module miteinander zu verbinden werden die Buchsen an ein Panel angeschlossen, welches eine direkte Pin-Verdrahtung durch eine Parallelschaltung aller Buchsen darstellt. Die synchrone Datenerfassung wird durch eine Master-Slave (Daisy-Chaining)-Struktur erreicht. Dazu wird ein Modul als Master definiert und steuert die restlichen Module als Slaves. Die Definition erfolgt innerhalb der DAQ-Software.
Neben der Synchronisation ist die Datenrate der Module zu beachten. Wie bereits beschrieben werden zur sicheren Stromversorgung, bis zu vier Modulen an einem USB-Hub und an einem USB-Port des PC-Host betrieben. Gemäß der USB 2.0 Spezifikationen wird die Datenrate an einer USB-Schnittstelle unter allen Teilnehmern aufgeteilt. Wenn die Datenrate der USB-Messgeräte höher ist, als die USB-Geschwindigkeit es zu lässt kann es zu einem Überlauf des FIFO-Buffer und damit verbunden zu einen Datenverlust kommen. Die theoretisch höchste Datenrate wird mit der höchsten Abtastrate von ca. 105 kHz erreicht. Die Gesamtdatenrate einer 16 kanaligen Messung beträgt

$$D = N_{CH} \cdot f_s \cdot N_{AD} = 16 \cdot 105\ kHz \cdot 24 = 40.320.000 Bits/s = 4,807 MByte/s \tag{3.4}$$

Die Datenraten des USB 2.0 Standard liegen bei 60 $MByte/s$ /54 $MByte/s$ /33 – 40 $MByte/s$ / (Brutto/Netto/real). Die ersten beiden Werte stellen das theoretische erreichbare Maximum der Übertragungsrate dar. Die Brutto-Datenrate berücksichtigt auch die Paket-Header, die Netto-Datenrate nur den reinen Inhalt. In der Praxis tauchen auch Fehler in der Übertragung auf, die durch den Übertragungsstandard korrigiert werden müssen. Dadurch sinkt die Netto- Übertragungsrate auf einen Wert von 33 – 40 $MByte/s$. Die maximal erreichbare Datenrate liegt bei ca. 5 $MByte/s$ und damit wesentlich unterhalb der nutzbaren Datenrate. Ein Betrieb von vier USB-Messgeräten mit einer Abtastrate von 105 kHz an einem USB-Port ist damit möglich.
Obwohl diese Datenrate die zur Verfügung stehende Kanalbandbreite nicht vollständig ausnutzt sind diese Datenraten vergleichsweise groß. Bei der Aufzeichnung von Signalen während einer Messung wird mit einer Datenrate von 4,807 $MByte/s$ pro Minute ein Festplattenspeicherbedarf von 288,42 $MByte$ benötigt für eine einstündige Messung entspricht das einem Platzbedarf von 16,9 $GByte$. So lässt sich feststellen, das die geplante Betriebsdauer des Messsystems von zehn Stunden mit 16 Mikrofonen bei einer Abtastrate von 48 kHz Daten mit einem Umfang von ca. 169 $GByte$ erzeugen.

ZUSAMMENFASSUNG

Die wesentlichen Eigenschaften des Module werden in der nachfolgenden Tabelle dargestellt.

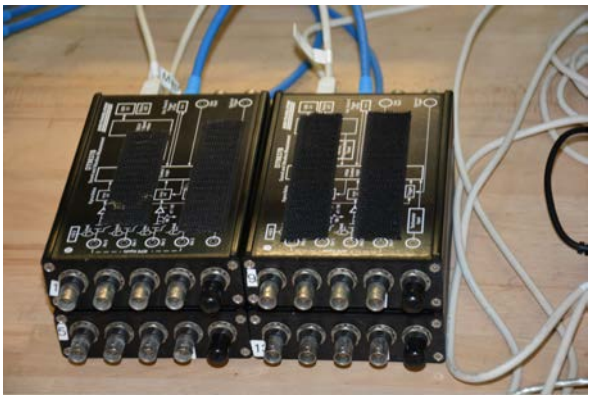
[3] Die Hot Swap-Funktion ermöglicht ein Abstecken und Anstecken ohne Beschädigung der USB-Module.

Tabelle 3.2: Zusammenfassung der wichtigsten Eigenschaften des DT9837B USB-Messgerät (gemäß Hersteller-Datenblatt [9])

	DT9837B
Anzahl der Eingänge	4 single-ended (BNC)
Amplitudenauflösung	24 Bit
Eingangsbereich	$\pm 1\ V$ bzw. $\pm 10\ V$
Konstantstromquelle	$4\ mA@18\ V$
Rauschen der Konstantstromquelle	$5\ nA(rms)$
Genauigkeit der Konstantstromquelle	$\pm 1\%$
Peak-Eingangsspannung	$\pm 40\ V$
Verstärkungsfehler	$\pm 0,02$ % bzw. $\pm 0,5$ %
Zero-Temperaturkoeffizient	$10\ \mu V/°C \cdot Gain\ +\ 100\mu\ V$
Verstärkungskoeffizient	$25 \cdot 10^{-3} °C$
Gruppenlaufzeit	$39 Samples(39/f_s)$
Frequenzgang ($-3\ dB$)	$0,1\ Hz - 0,49 \cdot f_s$
Sperrbanddämpfung	$-100\ dB$ bei $0,55 \cdot f_s$
Passband-Ripple	$\pm 0,005\ dB$
SNR	$106 dB$
THD ($-0,5\ dBFS$, $1\ kHz$ mit $f_s = 50\ kHz$)	$-92\ dB$ (typ.)
SFDR	$\leq -92\ dB$
Eingangsimpedanz	$1\ M\Omega$ mit $20\ pF$
Leistungsverbrauch	$\pm 0,3\ V@0,425\ A$
Abmessungen	$10\ cm; 19\ cm; 17,3\ cm;$
Gewicht	$491\ g$
zulässigen Betriebstemperaturbereich	$0 - 55°C$
zulässige Luftfeuchtigkeit	$\leq 95\%$
zulässigen Höhe	$\leq 3\ km$

Der modulare Aufbau des MAMS mit vier DT9837B-Modulen ermöglicht einen Mehrkanal-Messung von insgesamt 16 Kanälen. Die Module lassen sich kompakt arrangieren (siehe Abbildung 3.3). Vor die elektrischen Eigenschaften scheinen dem Zweck entsprechend angemessen zu sein. An dieser Stelle sei darauf verwiesen, das gemäß der Auswahl der Daten der Tabelle 3.2 ein erhöhtes Rauschverhalten erwartet wird. Das theoretische SNR eines 24 Bit A/D-Wandlers beträgt für ein voll ausgesteuertes Sinussignal theoretisch $SNR = 6 * 24 - 1,76 = 142,24 dB$. Data Translation gibt jedoch ein SNR von 106 dB an. Das um 40 dB geringere SNR lässt den Rückschluss auf ein erhöhtes Eigenrauschen zu. Ein SNR von 106 dB ist dennoch ein akzeptabler Wert.

Abbildung 3.3: Aufbau von vier DT9837B-Modulen im Labor



3.2.3. MESSKABEL

Die Messkabel stellen die physische Verbindung zwischen den Sensoren und der DAQ-Hardware dar. Durch die Verwendung von ICP-Mikrofonen mit der entsprechenden Signalkonditionierung sind die Anforderungen an die Messkabel geringer als bei der Verwendung von extern polarisierten Mikrofonen. Die Konstantstromübertragung ist relativ unempfindlich gegenüber äußeren Störgeräuschen.
Die DT9837B-Module und auch die ICP-Mikrofone besitzen jeweils einen BNC-Steckverbinder. Als Kabel wurden RG58-CU Koaxial-Kabel verwendet. Diese stellen einen guten Kompromiss zwischen den physikalischen/elektrischen Eigenschaften und den Kosten pro laufenden Meter dar. Die Bezeichnung RG-58 bezeichnet ein Kabeltyp mit definierten technischen Daten.
Der Aufbau des RG58-Kabels besteht aus einer Kupferlitze mit $19x0,18\ mm$ CU/SN-Kupferleitern. Als Dielektrikum wird ein PE Isolator mit einem Durchmesser von $2,95\ mm$ verwendet. Der Außenleiter (Schirm) ist als verzinntes Kupfergeflecht ausgeführt. Der Leiteraufbau wird durch einen PVC Mantel geschützt.
Der Kabelaufbau bestimmt die elektrischen und mechanischen Eigenschaften. Der Wellenwiederstand des Kabels beträgt $50\ \Omega \pm\ 3\Omega$. Ein Meter des RG58-Kabels besitzt eine Kabelkapazität von $103\ pF/m$ und einen Verkürzungsfaktor von $0,66$. Der Dämpfungsfaktor des Kabels ist für den Frequenzbereich der DAQ-Hardware bis maximal $50\ kHz$ nicht relevant. Bei $10\ MHz$ beträgt die Dämpfung auf $100\ m\ 4,7\ dB$.
Nach mehreren Vorbesprechungsterminen wurde die geplante Array-Geometrie von einem $20x20\ m$ Kreuz-Array zur einem Kreis-Array mit einem Radius von $15\ m$ geändert. Die kreisförmige Anordnung sollte mit 15 Mikrofonen realisiert werden. Um Infraschall-Signale aufzeichnen zu können sollte zusätzlich ein Infraschall-Mikrofon an das MAMS angeschlossen werden. Die Kabel für das Datenerfassungssystem wurden alle in Eigenleistung angefertigt. Insgesamt wurden 18 Messkabel mit einer Länge von jeweils $17\ m$ angefertigt. Zwei der Kabel sind als Reserve eingeplant gewesen, falls vor Ort ein Kabel beschädigt wird. Bei der Fertigung von Kabel ist beim Crimpen der Steckverbinder auf den Kabelenden besonders auf die auf die Trennung von Signalader und Masse zu achten. Sollte durch eine fehlerhafte Steckverbindung eine elektrische Verbindung zwischen Signalmasse und Signalführung entstehen wird dem Signal der Rauschanteil der äußeren Störsignale überlagert.
Der geplante Messort des Array lag an einem Abhang des Vulkangipfels mit einer Neigung von ca. 30° – 35°. Das Gelände ist dort sehr unwegsam und uneben. Die Installation vor Ort ist nur unter sehr großen Anstrengungen möglich. Aus Sicherheitsgründen ist die Installation des Arrays und die Verkabelung der Mikrofone mit der DAQ-Hardware nur mit einer Person möglich. Die Position des Mittelpunkts des Kreis-Arrays liegt ca. 30 m unterhalb der Hangkante. Um ein möglichst geringes Gewicht und eine geringe Kabellänge zu erzielen wird der Host-PC, zusammen mit den DT9837B-Modulen, im Mittelpunkt des Kreis-Arrays positioniert. Die Steuerung des Host-PC erfolgt mittels einer Remote-Desktop-Verbindung von einem zweiten Computer vom Hang aus. Die Remote-Verbindung wird mit einem CAT 6 LAN-Kabel mit Cross-over Adapter hergestellt. Dadurch kann auf eine Verlängerung der Kabel um 30 m zur Hangkante verzichtet werden. Das entspricht einer Einsparung von 480 m Kabel und einem Gewicht von 16,8kg.
Die Begrenzung der Kabellänge ist auch für die maximale Grenzfrequenz die durch die vorgesehene IEPE-Konditionierung bedingt ist relevant. Gemäß Gleichung (2.11) beträgt die maximale Grenzfrequenz

$$f_{max} = \frac{4\ mA}{2\pi \cdot 18\ V \cdot 103\ pF/m \cdot 17\ m} = 20.198,61\ Hz.$$

Mit den langen Kabel und der Konstantstromquelle des DT9837B-Moduls ist es möglich das

Messsystem bis zu einer Frequenz von etwa 20,2 kHz zu betrieben. Die Kabelkapazität eines 17 m langen RG58-Messkabels beträgt insgesamt 1,751 nF. Unter Berücksichtigung des Speisestroms von 4 mA kann das Signal in diesem Bereich auch voll ausgesteuert werden.
Neben den elektrischen sind auch die mechanischen Eigenschaften zu nennen. Der minimale Biegeradius des Kabels bezeichnet ein maximale zulässige Krümmung um die elektrischen Eigenschaften zu gewährleisten. Es beträgt für das Kabel $2,5cm$. Der Arbeitstemperaturbereich liegt zwischen $-20\ °C$ bis 70 $text\check{r}C$. Für die logistische Planung ist das Gewicht der Kabel vom großen Interesse. Das betrifft sowohl den Transport vom Dorf Stromboli zum Messort auf den Gipfel des Vulkansin 900 m Höhe, als auch den Transport mit dem Flugzeug im Frachtraum. Ein 1 km langes RG58-Kabel wiegt 35 kg (davon sind 18 kg Kupferanteil).

3.2.4. HOST-PC

Das MAMS ist ein PC-gestütztes Datenerfassungssystem mit vier USB-Messgeräten. Auf dem Host-PC wird die Messsoftware ausgeführt, welche die Steuerung der USB-Messgeräte übernimmt und die gemessenen Daten auf dem PC darstellt und speichert. Außerdem versorgt der PC mit seinen USB-Ports die DT9837B Module mit Strom.
Die Anforderungen an den Host-PC werden durch die unterschiedlichen Komponenten des Datenerfassungssystems bestimmt. Die DT9837B-Module benötigen einen PC mit folgenden Mindestanforderungen: Pentium 4 Prozessor, 1 GB Arbeitsspeicher, Bildschirmauflösung von 1024x768, Windows XP (32/64 Bit), Windows Vista, 7 oder 8 (32/64 Bit) und mindestens 4 GB freien Festplattenspeicher. Die Anforderungen der MATLAB-basierten Messsoftware sind ähnlich zu den oben genannten. Der Arbeitsspeicher muss jedoch mindestens 2 GB groß sein.
Von der letzten Forschungsreise zum Stromboli besteht noch eine Anschaffung eines Lenovo Thinkpad X230. Dieser leistungsfähige Computer wurde mit einem zusätzlichen Akku ausgerüstet und besitzt gut Eigenschaften für die bevorstehende Messaufgabe.

Tabelle 3.3: Zusammenfassung der Ausstattung des Host-PC (Lenovo Thinkpad X230)

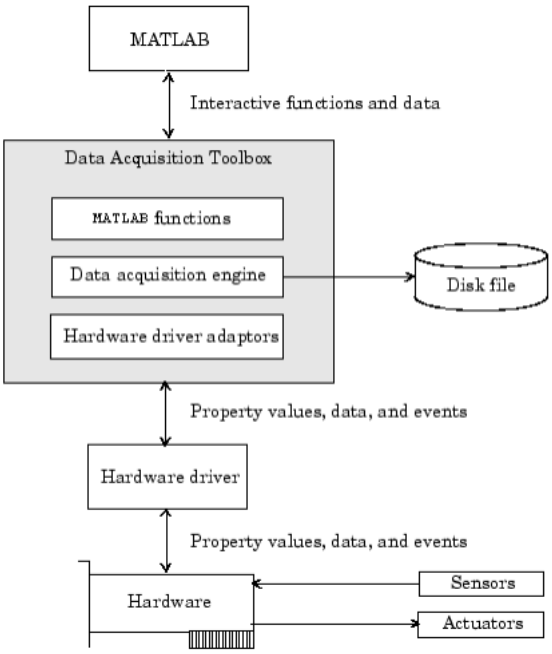
Bildschirm	
Auflösung	max. 1920 x 1200
Performance	
Prozessor	Intel Core i5-3220M
Arbeitsspeicher	8 GB
Festplattenspeicher	500 GB
Schnittstellen	
USB-Ports	2 x USB 3.0 / 1 x USB 2.0
LAN	1 x Gigabit Ethernet
Kommunikation	
Adapter	Centrino Wireless-N2200
Spannungsversorgung	
Intern	ThinkPad Battery 44+ (6 Zellen Akku, 63 Wh, 345 g)
Extern	ThinkPad Battery 19+ (6 Zellen Akku,64 Wh, 750 g)
Sonstiges	
Gewicht	1,7 kg
Betriebssystem	Windows 7 Professional (64-Bit)

3.2.5. MESSSSOFTWARE

Der Schwerpunkt der Entwicklung des Datenerfassungssystem lag in der Programmierung der Software zum steuern der DAQ-Module und zum Speichern der Messdaten. Außerdem mussten Informationen wie die Abtastrate, die Kanalkonfiguration an die Hardware gesendet werden. Die Softwa-

re muss zwischen der Treiber-Software und der Applikations-Software unterschieden werden. Der Software-Prototyp wurde mit MATLAB 2013a in einer 32-Bit Version entwickelt. Die Software wird unter Verwendung des Legacy-Interface der Data Acquisition Toolbox von Mathworks innerhalb der MATLAB Umgebung implementiert. Die DAQ-Toolbox besteht aus drei wesentlichen Komponenten (siehe Abbildung 3.4): den MATLAB-Funktionen, der Data Acquisition Engine und den Hardware Treibern.

Abbildung 3.4: Interaktionsdiagramm der Komponenten der DAQ-Hard- und Software unter Verwendung von MATLAB[7]



Der Hardware-Treiber erlaubt den Zugriff und die Kontrolle der Hardware. Grundlegende Funktionen sind zum Beispiel:

- das Lesen und Schreiben von Daten zwischen DAQ-Modul und Computer,
- Einstellen der Abtastrate,
- Integration der DAQ-Hardware und PC-Ressourcen,
- Integration der DAQ-Hardware und der Signalkonditionierung,
- Zugriff auf die Subsysteme der DAQ-Hardare und
- der Zugriff auf mehrere Instanzen der DAQ-Hardware.

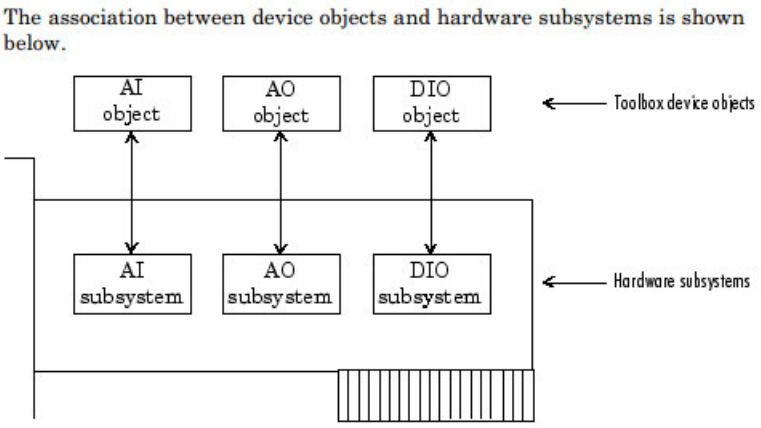
Die Treiber-Funktionen werden durch sogenannte DAQ-Adapter zur Verfügung gestellt. Für die verwendeten DAQ-Module stehen ebenfalls passende DAQ-Adapter der Firma Data Translation zur Verfügung. Die Applikations-Software stellt das Frontend des Messsystems für den Benutzer zur Verfügung. Es soll die Eingaben des Benutzers einlesen und das Signal aufbereiten, Events aus der Benutzer-Eingabe generieren und die erfassten Signale darstellen und speichern.
Die DAQ-Engine speichert Objekte und Property Values der Applikations-Software. Außerdem kontrolliert sie die Synchronisation der Events und die Speicherung bzw. Zwischenspeicherung der Daten. Die Engine und die Software laufen asynchron zu einander. Dadurch ist es möglich die Messung auszuführen und gleichzeitig das Frontend zu bedienen. Mit den MATLAB-Funktionen wird

es ermöglicht Objekte zu erzeugen, Daten zu erfassen, Die Konfiguration der DAQ-Hardware vorzunehmen und den Status der Datenerfassung und der Hardware zu evaluieren.
Die Erfassung der Daten bedeutet im Zusammenhang mit der DAQ-Engine die Daten von der DAQ-Hardware einzulesen und in die DAQ-Engine zu transferieren. Zum Datentransfer muss explizit eine getdata() Funktion auf gerufen werden. Wenn die Daten nicht extrahiert werden steigt der Speicherplatzbedarf an bis sein Maximum erreicht wird und ein DataMissed-Event erfolgt. Die Datenerfassung wird in diesem Moment automatisch beendet. Die Datenerfassung besteht aus zwei unabhängigen Schritten der Speicherung der Daten von der Hardware in die DAQ-Engine und die Extraktion der Daten aus der Engine in den Workspace bzw. auf die Festplatte.

ALLGEMEINER WORKFLOW DER DATENERFASSUNG

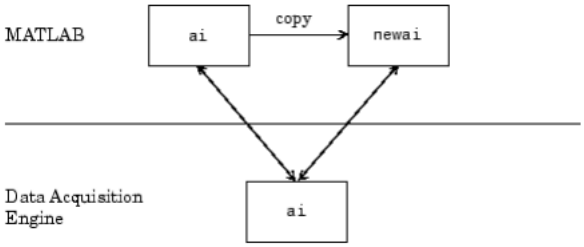
Innerhalb der DAQ Toolbox gibt es verschiedene Komponenten die erzeugt und gesteuert werden können. Unter ihnen gibt es sogenannte Objekte, die ein Interface zwischen der DAQ-Hardware und der Software-Applikation darstellt. Objekte werden Hardware-spezifischen Subsystemen zugeordnet. Wichtige Subsysteme sind analoge Input Subsystem oder analoge Output Subsysteme. Im allgemeinen Workflow wird zuerst ein Objekt im analogen Subsystem mit der Funktion analoginput() erzeugt. Die AI-Objekte stellen ein grundlegendes Element in der Datenerfassung dar. Anschließend werden die Kanäle hinzugefügt. Kanäle sind grundlegende Device Elemente. Ein Objekt kann nur Daten erfassen, wenn dem entsprechenden AI-Objekt auch mindestens ein Kanal zu gewiesen ist. Vor der eigentlichen Datenerfassung müssen die Eigenschaften des Objektes und der Kanäle konfiguriert werden. Das ist notwendig um das gewünschte Verhalten zu erhalten. Die meisten Eigenschaften können zu jeder Zeit konfiguriert werden, es gibt auch einige Eigenschaften die nur verändert werden können, wenn keine aktuelle Datenerfassung ausgeführt wird. Sobald das Setup der Datenerfassung eingestellt wurde kann die Messung mit dem Befehl start() begonnen werden. Die Datenerfassung erfolgt im Hintergrund während MATLAB weiterhin Befehl entgegennimmt. Die Daten die erfasst werden müssen aus der Engine extrahiert werden (wenn die Daten im Workspace zur Verfügung stehen sollen). Die Extraktion erfolgt mit der Funktion getdata(). Sobald die Datenerfassung beendet worden ist sollten die Objekte die nicht mehr gebraucht werden entfernt werden um notwendigen Speicher freizugeben.
Device Objekte sind Komponenten der DAQ Toolbox, welche auf die Hardware zu greifen können. Sie stellen so eine Verbindung zwischen der Funktionalität der Hardware zur Steuerung mit der Software her. Jedes Device Objekt ist mit einem spezifischen Hardware Subsystem assoziiert (siehe Abbildung 3.5).

Abbildung 3.5: Zusammenhang zwischen Device Objekt und Hardware Subsystem [7]



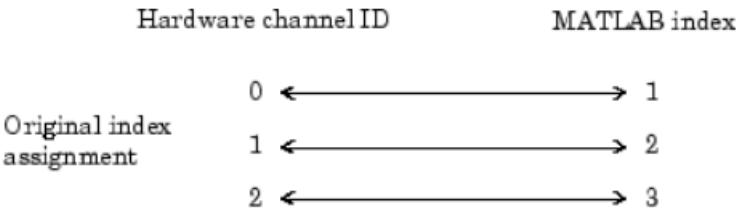
Um ein Objekt zu erzeugen muss eine *object creation function* aufgerufen werden, dafür muss auch der entsprechende Hardwaretreiber registriert sein. Die Erzeugung eines Device Objekt innerhalb von MATLAB erfolgt durch den Befehl-Aufruf analoginput(). Wenn Objekte erzeugt werden existieren sie sowohl in dem MATLAB Workspace, als auch in der DAQ-Engine. Durch dieses Funktionsprinzip ist es möglich einem Objekt in der DAQ-Engine mehrere aus dem Workspace zuzuweisen. Das ist zum Beispiel beim Kopieren eines AI-Objektes der Fall.

Abbildung 3.6: Zuordnung von Objekten im MATLAB Workspace und der DAQ-Engine [7]



Wenn das ursprüngliche oder kopierte AI-Objekt im Workspace gelöscht wird, wird auch das zugeordnete Objekt in der DAQ-Engine gelöscht.
Um eine Messung durchführen zu können muss dem AI-Objekt mindestens ein Eingangskanal zugeordnet werden. Das Hinzufügen von Kanälen zu einem Device Objekt erfolgt innerhalb der *Channel Group* mit dem Befehl addchannel().
Bei der Programmierung einer Anwendung mit MATLAB und der DAQ-Toolbox ist zu beachten, dass die Hardware Channel IDs numerische Werte, zur eindeutigen Identifikation sind, welche Nullbasiert sind. Das bedeutet sind fangen beim ersten Kanal mit 0 an. Die Arbeit mit Vektoren und Arrays ist jedoch Eins-basiert, das heißt das erste Element wird mit den Zugriff auf 1 abgerufen.

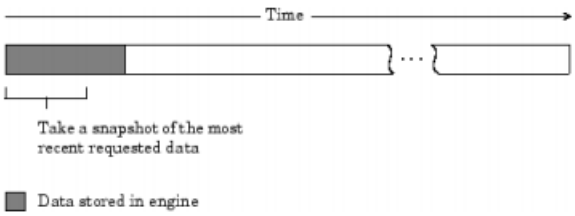
Abbildung 3.7: Hardware Channel IDs und Matlab Index[7]



Um ein Device Objekt bzw. die damit assoziierte Hardware zu konfigurieren müssen die Eigenschaften eingestellt werden. Es gibt verschiedene Eigenschaften die sich auf das gesamte Objekt (common properties) und Eigenschaften die sich auf den einzelnen Kanal eines Objektes (channel properties) beziehen. Jede dieser Eigenschaft besitzt allgemeine Eigenschaften (base properties) wie zum Beispiel die Abtastrate des AI-Subsystem, daneben existieren Geräte-spezifische Eigenschaften (device-specific properties) die nur für die entsprechenden Hardware-Geräte zur Verfügung stehen. Ein Beispiel sind die in Rahmen dieser Arbeit vorgestellten DT9837B-Module. Die Einstellung des Synchronisations-Modus ist eine geräte-spezifische Eigenschaft.
Die Objekte der Datenerfassung kennt zwei verschiedene Status: den Running-Status und den Logging Status. Der Running-Status bedeutet, dass das AI-Subsystem gerade Daten erfasst. Beim Logging-Status speichert das AI Subsystem Daten in der Engine oder auf dem Computer ab. Nach dem Start der Messung (mit dem Befehl start()) wird das Running Property automatisch auf On gesetzt und beide Objekte (Hardware und Device Objekt) arbeiten mit den zuvor eingestellten Eigenschaften. Während der Messung können die Daten mit dem Befehl peekdata() als Preview angeschaut wer-

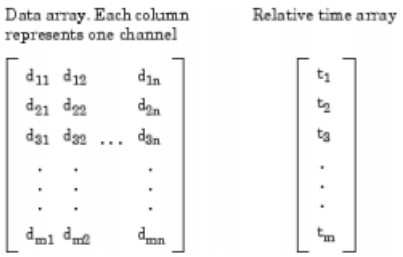
den. Die Daten werden mit dem peekdata() Befehl nicht aus der Engine extrahiert. Der Vorteil dieser Methode ist, das sie nicht priorisierend ist und deshalb auch die Datenerfassung nicht stört. Sie hat keinen Einfluss auf die Synchronisation der Datenerfassung. Während das Running Property des Objektes auf On steht können die erfassten Daten des AI-Subsystems im Arbeitsspeicher oder auf die Festplatte gespeichert werden. Wenn ein Objekt gestoppt wird, werden die Eigenschaften Running und Logging automatisch auf Off gesetzt.
Ein wichtiger Punkt für die Datenerfassung ist die Darstellung der gemessenen Signale im Frontend. So kann die Funktion der Mikrofone überprüft werden oder auftretende Fehler identifiziert werden. Um die Messdaten darzustellen ohne in den Prozess der Datenerfassung einzugreifen können nicht-blockierende Funktionen verwendet werden. Die Funktion peekdata() stellt Messdaten ohne Blockierung der Messung dar. Nachdem die Funktion ausgeführt wurde wird die Kontrolle augenblicklich an MATLAB zurück gegeben. Deshalb fehlen oder wiederholen sich möglicherweise einige Samples. Die peekdata()-Funktion gibt immer die aktuellsten Samples zurück (siehe Abbildung 3.8). Die Daten werden nicht aus der Engine extrahiert. Bei der Verwendung der peekdata()-Funktion

Abbildung 3.8: Extraktion von Messdaten aus der DAQ-Engine mit peekdata() [7]



muss beachtet werden, dass sie nur ausgeführt werden kann, wenn sich das Objekt im Messvorgang befindet (Running Property = On). Die Sample-Vorschau kann auch aufgerufen werden, wenn bisher noch kein Trigger ausgelöst wurde. Wenn die abzurufende Anzahl der Samples größer ist als die momentan vorhandene Anzahl an Samples, werden nur die bisher vorhanden Samples wiedergegeben zusammen mit einer Warnmeldung.
Die Extraktion der Daten von der Engine ist notwendig, weil es ansonsten zu einem Überlauf im Arbeitsspeicher kommen kann. Die Extraktion ist auch notwendig, wenn die Daten in einer Echtzeitanwendung verarbeitet werden soll. Durch die Daten-Extraktion mit dem Befehl getdata() wird ein Array erzeugt, welches neben den Messdaten auch Informationen über die relative Zeit, die absolute Zeit, Event- und Hardware-Informationen zurück gibt. Die Daten enthalten eine Spalte für jeden zugewiesenen Kanal der Hardware (siehe Abbildung 3.9). Im Gegensatz zu der zuvor erwähn-

Abbildung 3.9: Extraktion von Messdaten mit getdata() [7]



ten peekdata()-Funktion ist die getdata()-Funktion eine blockierende Funktion. Wenn die Funktion erfolgreich durchgeführt wird fehlen keine Samples und es gibt keine Wiederholungen. Es werden immer die ältesten Daten abgerufen.
Der Aufruf [data, time] = getdata(ai) gibt ein Array mit Daten (data) und einen Zeitvektor mit der

relativen Zeit (time) zurück. Der Zeitvektor ist mit den extrahierten Daten assoziiert. Die absolute Zeit ist auf den ersten Trigger bezogen.Die Trigger-Events bezeichnen Momente ab denen die Daten im Arbeitsspeicher oder auf der Festplatte geloggt werden.
Je nach Art des Triggers können verschiedene Typen von Trigger definiert werden. Es gibt drei wichtige Trigger-Typen: den Immediate-, den Manual- und den Software-Trigger. Der Immediate-Trigger wird sofort nach der Start-Funktion ausgeführt. Der Manual-Trigger wird nach einem manuellen trigger()-Aufruf ausgelöst. Als drittes gibt es den Software-Trigger der von einem digitalen TTL-Signal unter bestimmten Bedingung z.B. steigende oder fallende Flanke ausgelöst wird. Der Immediate-Trigger ist als Default-Typ voreingestellt. Eine kontinuierliche Datenerfassung kann mit einem Immediate-Trigger und der Einstellung der Samples pro Trigger auf unendlich (Inf) realisiert werden.
Die Leistungsfähigkeit einer DAQ-Applikation wird durch Verwendung von Events und Callbacks erhöht. Wenn bestimmte Ereignisse (Events) auftreten, können diese eine dazugehörige Callback-Funktionen aufrufen. Es gibt eine Vielzahl von Event-Typen die einen Callback auslösen können. Die zeitlichen Informationen zu den Ereignissen werden automatisch gespeichert.

Tabelle 3.4: Auswahl von Eventtypen und dazugehörige Callback-Funktionen

Eventtyp	Beschreibung	Callback-Funktion
Data Missed	Daten sind fehlerhaft	DataMissedFcn
Input Overrange	Eingangssignal hat den Eingangsbereich überschritten	InputOverRangeFcn
Runtime error	Runtime error wird ausgeführt, wenn ein Fehler in der Hardware oder ein Timeout vorliegt.	
Samples Acquired	Definierte Anzahl an zu erfassenden Samples wurden gemessen	SamplesAcquiredFcn
		SamplesAcquiredFcnCount
Start	Ausführung nach Aufruf der start()-Funktion	StartFcn
Stop	Ausführung nach Aufruf der stop()-Funktion	StopFcn

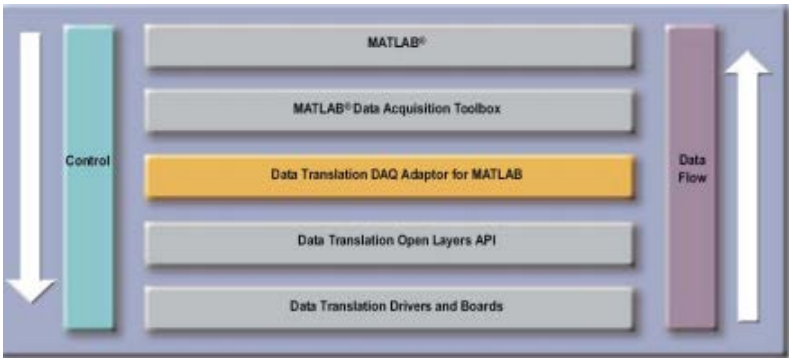
Bei der Ausführung der Callback-Funktionen ist zu beachten, dass sie in der Reihenfolge ihrer Aufrufe ausgeführt werden. Alle Aufrufe werden garantiert (bis auf die zeitbezogenen) ausgeführt. Die Callback-Funktionen können unter Umständen verzögert ausgeführt werden. Das ist unter Umständen der Fall, wenn CPU intensive Aufgaben wie z.B. die Darstellung von Grafiken durchführt.

AUFBAU DER MAMS-SOFTWARE-APPLIKATION

Die Software-Applikation wurde innerhalb von MATLAB 8.1 (R2013a) in einer 32-Bit Version implementiert. Die Datenerfassung wird von der MATLAB Data Acquisition Toolbox 3.3 und Data Translation Open Layer Adaptor unterstützt. Die nachfolgend vorgestellte Applikation wurde unter Verwendung des DTOL-Adapter in der Version 1.0.10.16 erstellt.
Der DAQ-Adapter ist ein Interface zwischen MATLAB, der DAQ-Toolbox und der Data Translation Hardware dar. Dazu stellt der Adapter eine API zu allen analogen Input Modulen zur Verfügung. Mit der DAQ-Toolbox in Kombination mit der Data Translation Open Layer-Struktur ist es möglich Daten mit MATLAB zu analysieren und darzustellen, die mit einer Data Translation Hardware gemessen wurde. Die Funktionen der Data Translation Open Layer API stellen Funktionen zur Erzeugung von Subsystemen und Kanälen, sowie die Erfassung von Daten zur Verfügung. Die Abbildung 3.10 zeigt die hierarchische Kommunikationsstruktur der verschiedenen Programmebenen von MATLAB und den Data Translation Komponenten.

Die gesamte Applikation ist als Callback-basierte GUI implementiert. Das bedeutet die Software reagiert auf bestimmte Ereignisse. Das Drücken einer Taste löst z.B. die KeyPressFcn aus, wel-

Abbildung 3.10: Datenflussmodell zwischen Data Translation Open Layer Struktur und MATLAB [9]



che wiederum definierte Aktion ausführt. Ein wesentlicher Punkt der Datenerfassung mit Mikrofonen ist die Möglichkeit der Pegelkalibrierung der Messkette. Durch die Umwelteinflüsse und der Komponenten der Messkette im Feld (z.B. Messkabel) kann der absolute Pegel etwas höher oder niedriger liegen als bei ein kalibrierten Mikrofon unter Laborbedingungen. Daher muss die gesamte Messkette kalibriert werden. Üblicherweise erfolgt die Pegelkalibrierung durch Bestimmung des Mikrofonübertragungsfaktor während der Einsatzsituation. Weicht der gemessenen Schalldruckpegel vom Kalibrierpegel ab muss ein neuer Übertragungsfaktor bestimmt werden. Die Applikationssoftware unterstützt eine Kalibrierung von Mikrofonen und speichert die zur späteren Analyse der Daten ab.
Die Software umfasst einen Ordner mit voreingestellten Konfigurations-Dateien der sogenannten Mikrofon-Datenbank (microphoneDatabase), einer .mat-Datei mit der letzten verwendeten Mikrofon-Konfiguration (mircophoneData.m), 22 .m-Dateien und dem ausführendem Skript MAMS_GUI_v8.m.
Die Initialisierung der GUI wird mit dem MATLAB-Skript MAMS_GUI_v8.m durchgeführt. Dieses Skript konfiguriert die angeschlossenen DT9837B-Module, die Fenster zu Darstellung und ein Struktur mit den Mess- und Benutzerdaten. Der Ablauf erfolgt im wesentlichen mit folgenden Schritten

- Löschen aller alten Daten,
- Initialisierung der Haupt-GUI,
- Auswahl des Mikrofon-Setups,
- Initialisierung der DAQ-Module,
- Initialisierung der Subplots und
- Definition der UserData-Struktur.

Nachdem das Skript ausgeführt wurde ist das Messsystem bereit zum Einsatz. Die weitere Interaktion mit der GUI erfolgt durch zuvor definierte Callback-Funktion und somit Ereignis-basiert.
Das Löschen alter Daten und Objekte betrifft vor allem die DAQ-Engine und den MATLAB Workspace. Mit dem Befehl *daqreset* werden die geladenen DAQ-Adapter und Objekte aus der DAQ-Engine und dem MATLAB Workspace entfernt. Das ist in sofern notwendig um möglich Konflikte mit den neuen zu initialisierenden Objekten zu vermeiden und die PC-Ressourcen zu schonen.

INITIALISIERUNG DER HAUPT-GUI

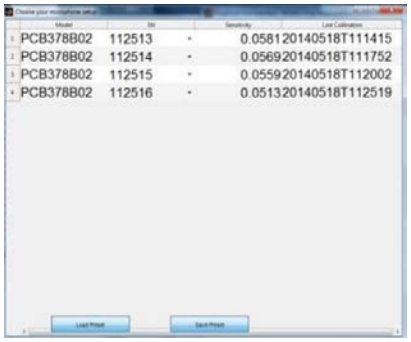
Die erste Initialisierung der Haup-GUI erzeugt eine Figure die später alle Elemente enthält, mit der die Messung gesteuert werden kann. Bei der Erzeugung des Haupt-GUI werden bereits zwei Callback-Funktionen an die GUI übergeben.

```
1  hMamsGui     = figure(... % Definition des Main-GUI Fensters
2      'Units'              , 'pixels'                        ,...
3      'Color'              , btnColor                        ,...
4      'Colormap'           , []                              ,...
5      'DeleteFcn'          , 'close_callback(gcbf)'          ,...
6      'HandleVisibility'   , 'on'                            ,...
7      'KeyPressFcn'        , 'keyEvent_callback(gcbf)'       ,...
8      'MenuBar'            , 'none'                          ,...
9      'Name'               , ['Graphical User Interface -'   ,...
10     ' Mobile Array Measurement System (MAMS)']             ,...
11     'NumberTitle'        , 'off'                           ,...
12     'Pointer'            , 'arrow'                         ,...
13     'Position'           , figPos                          ,...
14     'Tag'                , 'Main-GUI'                      ,...
15     'UserData'           , []                              ,...
16     'Visible'            , 'On');
```

In Zeile 5 wird eine Delete Funktion (DeleteFcn) definiert. Diese wird immer ausgeführt, wenn das Fenster geschlossen wird. Das Übergabeargument *gcbf* ist eine MATLAB interne Funktion und verweist auf das Handle der Figure welches die Objekte enthält und den Callback aufgerufen hat. Der Aufruf der Funktion löscht das Handle und die damit verbunden Daten aus dem Workspace.
Die Key Press Funktion (KeyPressFcn) ist eine der wichtigsten Komponenten der Software-Applikation. Sie stellt die Schnittstelle zwischen dem Benutzer und der Messteuerung des Messsystems dar. Der Callback-Aufruf dieser Funktion erfolgt immer, wenn der Benutzer eine Taste auf der PC-Tastatur drückt. Um eine möglichst schnelle und einfache Messung durchzuführen wird die Messung ausschließlich mit der PC-Tastatur gestartet und gestoppt. Ein Druck auf die Leertaste startet die Messung beendet diese ebenfalls. Der Callback-Funktion wird das Handle zur Figure der Haupt-GUI übergeben. Die wesentlichen Parameter der Key Press Funktion sind die Felder UserData und CurrentCharacter des Figure Handle. Das Feld CurrentCharacter beinhaltet einen String mit dem zuletzt eingegebenen Zeichen der Tastatur. Das Feld userData enthält alle Handles und Variablen die für die Messung benötigt werden. Es stellt den Kern der Applikation dar. Durch die Übergabe der Verknüpfung der Benutzerdaten mit dem UserData Feld der HauptGUI ist den Funktionen ein übergreifender Zugriff auf alle Parameter der Applikation von jeder Funktion aus möglich.
Wenn bei Übergabe der Status (userData.daqState) den String 'Start' beinhaltet und die aktuelle Tastatur-Eingabe ein Leerzeichen ist wird die Messung gestartet. Zu Beginn der folgenden Routine werden die Mikrofondaten aus der aktuellen Konfigurationsdatei eingelesen. Anschließend wird in Abhängigkeit der Anzahl der angeschlossenen und konfigurierten DAQ-Module für jedes Modul der Syncronisations-Modus eingestellt und der Speicherort der zu erfassenden Daten mit einem Zeitstempel festgelegt. Der Zeitstempel wird nach UTC angegeben. Für das Post-Processing wird die verwendete Konfigurationsdatei der Mikrofone ebenfalls auf der Festplatte gespeichert. Anschließend wird die Messung gestartet. Bei einer Messung mit mehr als einem Modul werden die Slave-Module vor dem Master-Modul gestartet. Die Datenerfassung beginnt erst, sobald das Master-Module den Startbefehl erhalten hat und diesen an die Slave-Module weiter gegeben hat.
Im Falle einer bereits laufenden Messung und der Eingabe eines Leerzeichen wird der aktuelle Messvorgang beendet. Das Master-Module wird zuerst gestoppt und anschließend die Slave-Module. Die Applikation ist anschließend bereit mit einer weiteren Messung zu beginnen.
Sollte eine Messung bei Aufruf der KeyPressFcn bereits durchgeführt werden und eine andere Eingabe als ein Leerzeichen erhalten wird diese Tastatureingabe mit einem Zeitstempel in einer Log-Datei auf der Festplatte gespeichert. Das Event-Log wird als .mat-Datei hinterlegt. Die Veränderung an der userData-Struktur werden zum Ende der Routine an das Feld UserData der Figure-Handle der Haupt-GUI zurück gegeben.
Im Gegensatz zu dem klassischen Weg der Datenspeicherung im Arbeitsspeicher und auf der Fest-

platte (LoggingMode = 'Disk and Memory') benutzt die MAMS-Applikation lediglich den Festplattenspeicher (LoggingMode = 'Disk'). Dadurch wird der Arbeitsspeicher geschont und steht der Zwischenspeicherung zu Verfügung. Die Datenabfrage und -darstellung erfolgt anhand der Daten die mit peekdata() abgerufen werden. Dadurch wird der Datenerfassungsprozess nicht gestoppt und es kann eine schnelle Darstellung der Ergebnisse erfolgen. Bei den dargestellten Signalen können einige Samples fehlen, jedoch gilt das nicht für die gespeicherten Signale auf der Festplatte. Es handelt sich hier um zwei verschiedene Prozesse. Die Visualisierung stellt in der jetzte Ausbaustufe der Applikation lediglich ein Instrument zur Überwachung der Funktionsfähigkeit der Module und ihrer einzelnen Kanäle dar. Nachdem die Haupt-GUI vorinitialisiert wurde, wird dem Benutzer die Möglichkeit gegeben die aktuelle Konfiguration des Messsetups festzulegen.Dazu erfolgt der Funktionsaufruf editMicrophoneData(). Es wird ein neues Fenster mit den bisher aktuell hinterlegten Mikrofonsdaten geöffnet. Anschließend kann durch die Auswahl der Mikrofone die Konfiguration geändert werden. Das Mikrofon-Editiert-Fenster untersützt auch das Laden und Speichern von Mikrofon Presets.

Abbildung 3.11: Auswahlfenster der editMicrophoneData()-Funktion.

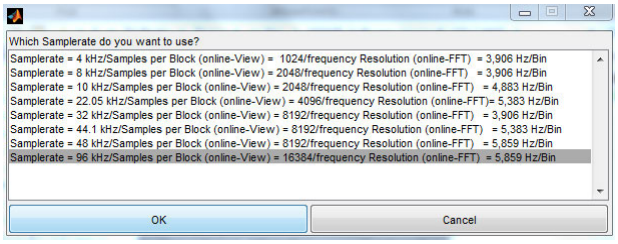


Neben dem Modelname und der Seriennummer wird auch der letzte verwendete Mikrofon-Übertragungsfaktor und der dazugehörige Zeitstempel (UTC) angezeigt.
Nach Einstellen der gewünschten Konfiguration des Messsetups wird die Initialisierung der angeschlossenen DAQ-Module entsprechend den zuvor getroffenen Angaben ausgeführt.

```
1 % Intialisierung der Module
2 %Definition der Eingabe-Struktur zur Intialisierung der DAQ-Module
3 daqSetupInput.hInfoText      = hInfoText;
4 daqSetupInput.hMamsGui       = hMamsGui;
5 %Funktionsaufruf zur Intialisierung der DAQ-Module
6 daqSetup                     = initDaqSetup(daqSetupInput);
```

Die Funktion initDaqSetup erzeugt nach dem Aufruf eine Fenster indem der Benutzer die gewünschte Abtastrate des Systems einstellen kann. Die Abtastfrequenz des Systems beschreibt die Anzahl der

Abbildung 3.12: Auswahl der gewünschten Abtastrate der aktuellen MAMS-Messung.



Stützstellen mit der das Signal pro Sekunde abgetastet wird. Die Messergebnisse können bis maxi-

mal der halben Abtastrate ausgewertet werden, ohne Aliasing-Artefakte zu enthalten. Zur Darstellung des gemessenen Signale kann die Größe der Datenblöcke gewählt. Diese Länge eines Blocks hat keine direkten Auswirkungen auf Messergebnisse, sondern nur auf die Darstellung der Daten und bestimmt die Frequenzauflösung der Echtzeit-Darstellung einer FFT. Eine Darstellung der spektralen Komponenten erfolgt bisher nur bei der Kalibrierung der Mikrofone. Die Größe der Signalblöcke variiert je nach Abtastrate (4 kHz – 96 kHz) von 1024 bis 16384 Samples pro Block. Die Auflösung (Bin-Abstand) der FFTs beträgt demnach etwa 4 Hz bis 6 Hz.
Anschließend wird der Benutzer gefragt welches der Module als Master-Modul verwendet werden soll. Die übrigen Module werden automatisch als Slave definiert. Der Unterschied zwischen Master und Slave liegt in der Funktion des Synchronisations-Port. Ist das Modul als Master definiert, so sendet der Port Befehle. Ein Slave kann Befehle an RJ45-Port nur empfangen, selber jedoch keine senden. Beim Ausführen de Applikation werden die Messungen der Slave-Module zuerst gestartet. Aufgrund ihrer Synchronisations-Einstellung beginnen sie erst mit der Messung, wenn sie den dazugehörigen Befehl vom Master-Modul erhalten haben.

INITIALISIERUNG DER DAQ-MODULE

Mit der Funktion initdaqsetup() werden in erster Linie die USB-Messgeräte und die dazugehörigen Kanäle konfiguriert.

```
1  set([master,slave1,slave2,slave3],...
2              'BufferingConfig'        , [blockLengthDAQ, lengthBuffer],...
3              'SamplesAcquiredFcnCount', blockLengthDAQ,...
4              'SamplesPerTrigger'      , Inf           ,...
5              'SampleRate'             , sampleRate    ,...
6              'LogToDiskMode'          , 'Index'       ,...
7              'LoggingMode'            , 'Disk'        ,...
8              'TimeOut'                , 2             ,...
9              'TriggerRepeat'          , 0             ,...
10             'TriggerType'            , 'Immediate'   );
```

Die obige Liste zeigt eine Konfigurations-Beispiel für insgesamt vier Module (master, slave 1, slave2, slave 3). Der Parameter BufferingConfig spezifiziert den Speicherplatzbedarf, der für die Module alloziert werden soll und als Zwischenspeicher dient. Er wird durch die Länge eines Buffer-Blocks (blockLengthDAQ) und der Anzahl der Blöcke (lengthBuffer) definiert. Mit der Einstellung des Parameters SamplesPerTriggers gleich Inf (unendlich) wird die Datenerfassung zu einer kontinuierlichen Datenerfassung gezwungen. Die Messung beginnt sofort nach Aufruf der start()-Funktion (TriggerType Immediate). Eine Wiederholung des Triggers ist nicht vorgesehen Die Messungen erfolgt mit der vom Benutzer angegebenen Abtastrate. Die Speicherung der Daten erfolgt ausschließlich auf der Festplatte um Resourcen des Arbeitsspeichers zu sparen. Die oben aufgeführten Code-Zeilen enthalten auch einen Callback-bezogenen Parameter. Der SamplesAcuiredFcnCount spezifiziert die Anzahl der Samples ab wann die dazugehörige SamplesAcquiredFcn ausgeführt werden soll. Diese wird im Anschluss definiert.

```
1    set(master,...
2              'SamplesAcquiredFcn'        , {@plotData1,aInput.hMamsGui},...
3              'LogFileName'               , ...
4                  fullfile(folderStr,'DAQSession_Master',userDateStr));
```

Der Programmzeilen in Zeile 2 definieren die Funktion plotData1() als Callback-Funktion für die SamplesAcquieredFcn. Der Funktion wird zusätzlich das Handle zur Haupt-GUI übergeben, um so mit Funktionsaufruf auch auf die Daten der Modul zugreifen zu können. Die PlotData1-Funktion

stellt den aktuellen Signal-Block in der Haupt-GUI dar und aktualisiert die Darstellung des bisherigen Signalverlaufs. Der LogFileName beschreibt den Ort und den Name unter dem die Daten als .daq-Datei abgespeichert werden.
Die Konfiguration der Kanäle eines Moduls erfolgt für jedes der angeschlossenen Module mit der Funktion configureDT9837BChannels(). Um die ICP-Mikrofone für die Messung verwenden zu können müssen die Kanäle auch entsprechend konfiguriert werden. Die Konfiguration wird durch setzten der Kanaleigenschaften durchgeführt.

```
set(aInput.module.channel(get(tmpChan,'Index')),...
                 'ExcitationCurrentSource'   ,'Internal',...
                 'Coupling'                  ,'AC',...
                 'GainPerChan'               , 1,...
                 'ChannelName'               , ...
                 ['Ch',num2str(aInput.n,'%02d')]);
```

Die ICP-Mikrofone benötigen eine Konstantstromquelle um richtig betrieben werden zu können. Die interne Stromquelle der DT8937B-Module wird durch den Parameter ExcitationCurrentSource dargestellt. Sobald er den Wert 'Internal' besitzt wird die Stromquelle angeschaltet. Um die Arbeitspunktspannung des Eingangssignal von der Wechselspannung des Messsignals zu trennen muss das Signal mit einem Hochpassfilter gefiltert werden. Mit der Einstellung Coupling = 'AC' wird der Hochpassfilter mit einer Grenzfrequenz von $-3\ dB$ in den Signalweg gesetzt. Die Kanalverstärkung (GainPerChain = 1) stellt den Messbereich auf einen Bereich von $-10\ V$ bis $+10\ V$ ein.

INITIALISIERUNG DER SUBPLOTS

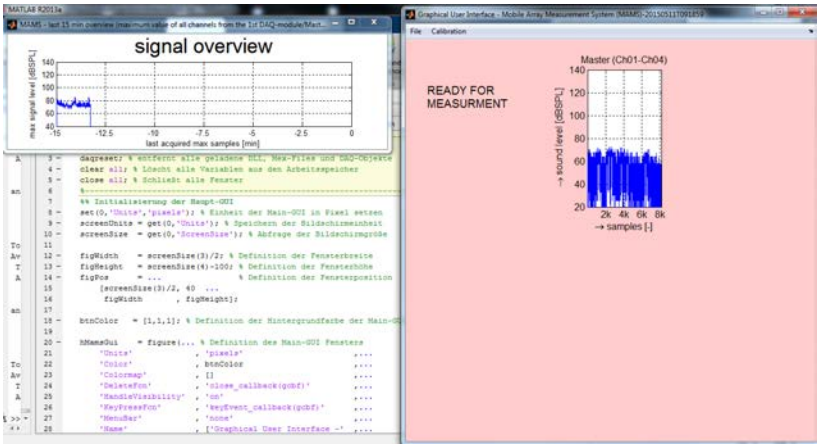
Mit den folgenden Programmzeilen werden die Subplots in der Haupt-GUI erzeugt.

```
    %% Intialisierung der Module
    % Definition der Eingabe-Struktur zur Intialisierung der DAQ-Module
    daqSetupInput.hInfoText     = hInfoText;
    daqSetupInput.hMamsGui      = hMamsGui;
    % Funktionsaufruf zur Intialisierung der DAQ-Module
    daqSetup                    = initDaqSetup(daqSetupInput);
```

Die Haupt-GUI beinhaltet vier Subplot zur Darstellung des jeweils aktuellen Signalblocks. Die Signale werden über einen Pegelbereich von 20 dB bis 140 dB dargestellt. Wenn weniger als vier Module bei der Messung verwendet werden, werden die Signalfenster der nicht verwendeten Module ausgeblendet.
Nachdem die Subplots intialisiert wurden, wird eine neues Fenster erzeugt. Es stellt den Signalverlauf der letzten 15 Minuten der Messung dar. Es zeigt jeweils das, über alle Kanäle des Master-Moduls, ermittelte Maximum in diesem Fenster an.
Im nächsten Schritt werden die verschiedene Menü-Elemente der Haupt-GUI erzeugt. Diese sind in zwei Menüs mit verschiedenen Menüpunkten gegliedert. Es gibt die Menüs File und Calibration. Im Menü File kann mit Klick auf den Menüpunkt Close die Callback-Funktion close_callback() aufgerufen und die Haupt-GUI damit geschlossen werden. Im Menü Calibration stehen drei Möglichkeiten zur Verfügung. In diesem Menüpunkt können die aktuell hinterlegten Mikrofondaten betrachtet (view micropohone data), die aktuellen Mikrofondaten bearbeitet (edit microphone data) und die Mikrofone kalibriert werden (calibrate microphone). Jeder Menüpunkt ruft eine andere Callback-Funktion auf.

Zum editieren des Mikrofon-Setups wird die editMicrophoneData()-Funktion aufgerufen. Zum Betrachten der aktuellen Konfiguration der Mikrofone wird die displayGuiMicInfo()-Funktion ausgeführt. Die Mikrofon-Daten werden in einer Tabelle dargestellt.

Abbildung 3.13: Darstellung der MAMS-GUI und der Signalverlaufs-GUI nachdem eine Messung beendet wurde.



DATEN-STRUKTUR

Zum jetzigen Zeitpunkt wurden die DAQ-Module initialisiert und die dazugehörigen Eingangskanäle konfiguriert. Außerdem wurde eine Haupt-GUI und eine GUI zur Darstellung des bisherigen Messsignals erzeugt. Die Haupt-GUI enthält ein Signal-Fenster mit dem aktuellen Signalblock für jedes aktive USB-Messgerät.
Die Interaktion der GUI mit den Benutzereingaben basiert auf eine Ereignis-abhängigen Callback-Steuerung. Damit bei einem Callback Aufruf alle notwendigen Informationen zur Verfügung stehen werden die Daten aller Fenster, Module und Konfigurationseinstellungen mit dem Handle der Haupt-GUI verknüpft. Dazu wird eine userData-Struktur angelegt und im Handle unter dem gleichnamigen Feld userData gespeichert. Die Callback-Funktionen werden oft mit dem Argument gcbf aufgerufen. Der Ausgabewert der gcbf Funktion gibt das Handle vom Objekt zurück, welches die Callback-Funktion aufgerufen hat. Das Handle verweist wiederum auf die Haupt-GUI, welche die userData-Struktur enthält. Dadurch ist eine funktions-übergreifender Austausch der Daten möglich.
Die Ergebnisse der Messung werden auf der Festplatte des Host-PC gespeichert. Die notwendige Ordnerstruktur wird automatisch durch die Applikation erzeugt. Für jeden Tag wird ein Ordner erzeugt, welcher wiederum folgende Unterordner enthält

- DAQSession_Calibration
- DAQSession_LogData
- DAQSession_Master
- DAQSession_MicrophoneData
- DAQSession_Slave1
- DAQSession_Slave2

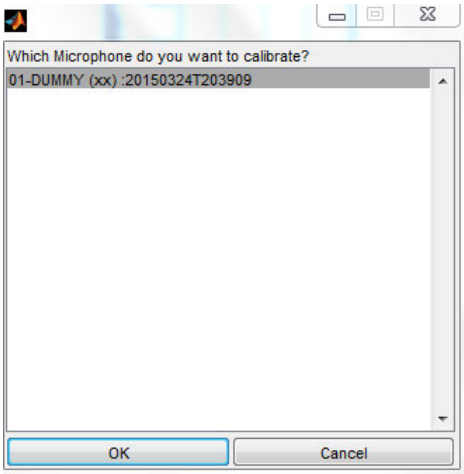
- DAQSession_Slave3

Der Ordner DAQSession_Calibration enthält die Zeitsignale während der Kalibrierung des jeweiligen Kanals im Format Calibration_Channel_X_Y.daq. X ist eine zweistellige Zahl, die den Messkanal (1 – 16) spezifiziert. Das Datum und die Uhrzeit werden an der Stelle Y kodiert z.b. wird der 01.01.1990 12:34:56 Uhr als 19900101T123456 dargestellt. Wenn der Kanal um diese Uhrzeit kalibriert wurde lautet die dazugehörige Kalibrierdatei Calibration_Channel_01_19900101T123456.daq. Wenn kein Mikrofon während der akutellen Messung kalibriert wurde ist der Dateiordner leer.
Im LogData-Ordner werden die jeweiligen Event-Dateien hinterlegt. Die Event-Daten werden jeweils als .mat-Datei abgespeichert und besitzen das Messdatum als Dateinamen z.B. 19900101T123456.mat. Dasselbe gilt auch für den MicrophoneData-Ordner. Bei jedem Messstart wird eine Datei erzeugt, die eine akutelle Konfiguration der Mikrofone und die dazugehörigen Informationen z.B. der letzten Kalibrierung und dem dazugehörigen Mikrofonübertragungsfaktor enthält.

KALIBRIERUNG VON MIKROFONEN

Die Pegelkalibrierung ist ein wichtiger Vorgang um die absolute Empfindlichkeit des Messmikrofons zu bestimmen. Abweichungen der Messstrecke z.B. durch die angeschlossenen Messkabel werden dadurch mitberücksichtigt. Die Kalibrierung wird unter Verwendung eines akustischen Kalibrators durchgeführt. Dieser erzeugt einen Referenzton um diesen mit den Messmikrofonen zu messen. Der Kalibrierton liegt in der Regel bei einer Frequenz von $f = 1\ kHz$ und besitzt einen Pegel von 94 $dB\ re\ 20\ \mu Pa$[4]. Mit der Kalibrierung wird die Mikrofonempfindlichkeit justiert. Die Messung des Pegels des Referenztons ermöglicht eine schnelle Überprüfung der Funktionsfähigkeit des Messsystems. Die Software-Applikation stellt die Möglichkeit der Kalibrierung zur Verfügung.
Über den Menüpunkt Calibration - calibrate microphone wird die Routine zur Mikrofon-Kalibrierung (calibrateMic.m) gestartet. Im ersten Schritt wird der Benutzer gefragt welches Mikrofon kalibriert werden soll.

Abbildung 3.14: Auswahlfenster des zu kalibrierenden Mikrofons der MAMS-Software-Applikation.



Nach Auswahl des Mikrofons und der Bestätigung das der Vorgang durchgeführt werden soll wird ein Fenster geöffnet, welches eine Pegel-korrekte FFT im Bereich von 500 Hz bis 2 kHz. Für die Kalibrierung wurde die SamplesAcquiredFcn geändert. Nachdem ein Signalblock erfasst wurde, wird die getSensitivity-Funktion aufgerufen.

[4] in Umgebungen mit einem hohen Umgebungsgeräuschpegel werden auch höhere Pegel z.B. 114 dB re 20 µPa verwendet.

Der Algorithmus basiert auf den Grundlagen der Frequenzanalyse. Der aktuelle Signalblock wird zuerst mit einer Fenster-Funktion gewichtet. Da es sich bei der Pegelkalibierung um eine einzelnen Ton (i.d.R. 1 kHz) handelt und die Genauigkeit der Amplitude sehr wichtig ist, wird das Eingangssignal mit einem Flat-Top-Fenster gewichtet. Dieses Fenster hat eine sehr hohe Amplitudengenauigkeit, der Amplitudenfehler ist praktisch Null [8]. Die Frequenzauflösung des Flattop-Fenster ist jedoch relativ schlecht, weil für jede Frequenzkomponente etwa 9-10 weitere Linien im Spektrum abgebildet werden [8]. Bei der Pegelkalibrierung ist jedoch die genaue Analyse der Amplitude wichtiger als die Kenntnis über die korrekte Signalfrequenz, solange diese innerhalb eines zulässigen Frequenzbereich von 960 Hz bis 1040 Hz liegt. Damit der Pegel richtig abgebildet werden kann muss das gewichtete Signal je nach Wahl der Fenster-Funktion skaliert werden. Der Skalierungsfaktor entspricht dem Kehrwert des Summe der Gewichtungsfaktoren des Fensters normiert auf die Länge der Folge.
Als erstes wird das komplexe Spektrum des Eingangssignal nach [31] berechnet. Die Berechnung der DFT mithilfe des FFT-Algorithmus liefert ein komplexes Zwei-Seitenspektrum S_{AA}. Jeder dieser spektralen Hälften besitzt jeweils die Halbe Energie des gesamten Spektrums. Die negativen Frequenzen sind daher redundant, weil sie das symmetrische Abbild der positive Frequenzhälfte darstellt und jeweils die Hälfte der Energie besitzt. Daher ist es möglich den redundanten Teil des Spektrum zu vernachlässigen und so ein Einseiten-Spektrum G_{AA} zu berechnen. Damit die Energie des Signals gleich bleibt muss die positive Frequenzhälfte mit Zwei multipliziert werden. Das gilt jedoch nicht für den Gleichspannungsanteil (f= 0 Hz), weil das Spektrum symmetrisch um dieses Bin ist.

$$G_{AA}(i) = S_{AA}(i), i = 0(DC) \tag{3.5}$$

$$G_{AA}(i) = 2 \cdot S_{AA}(i), i = 1 - \frac{N}{2} - 1 \tag{3.6}$$

Die Frequenzbins von $N/2 - N - 1$ werden nicht berücksichtigt. Die Höhe der Amplitudenwerte der einzelnen Frequenzkomponenten beträgt jeweils

$$\frac{A_k^2}{2} = (\frac{A_k^2}{\sqrt{(2)}}), \tag{3.7}$$

wobei $\frac{A_k}{\sqrt{(2)}}$ dem Effektivwert eines Sinus-Signals entspricht. Um das Amplitudenspektrum aus der komplexen FFT zu erhalten muss es noch entsprechend konvertiert und skaliert werden [30].

$$Amplitude[V_{pk}] = \frac{Magnitude}{N} = \frac{\sqrt{real(FFT(A))^2 + imag(FFT(A))^2}}{N} \tag{3.8}$$

Die Komponenten der Spektrallinien stellen jeweils den Spitzenwert der Frequenzkomponente dar. Zur Umwandlung in einen Effektivwert ist das Zweiseitenspektrum durch den Crest-Faktor für Sinus-Signale zu dividieren. Mit der Multiplikation von 2 aufgrund des Wegfalls der redundanten Frequenzkomponenten.

$$Amplituden[V_{rms}] = \frac{Amplitude[V_{pk}]}{\sqrt{2}} \tag{3.9}$$

Die Gleichung (3.9) stellt die pegelrichtige FFT dar. Nur bei der korrekten Berechnung der FFT und richtiger Anwendung der Fensterung ist es möglich die Pegelkalibrierung durchzuführen. Die Implementierung des Kalibrier-Algorithmus in MATLAB ist im folgenden aufgeführt.

```
1   %Kalibrierungs-Algorithmus-------------------------------------------
2   % Fensterung inklusive Amplituden-Korrektur+++++++++++++++++++++++++++
```

```
3        % 1. Berechnung des Gewichtungs-Fenster für die Fensterung des
4        % erfassten Messdaten-Block
5        winFFT          = flattopwin(length(data(:,channelIdx)));
6        % 2. Fensterung des aktuellen Messdaten-Block
7        sigWin          = data.*winFFT*(length(data)/sum(winFFT));
8        % length(data)/sum(winFFT) ist Amplitudenkorrektur für die
9        % Energieänderung durch das Fenster
10       %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
11       % 3. Berechnung der komplexen Fourier-Transformieten+++++++++++++++++
12       dataComplexFFT  = fft(sigWin);
13       %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
14       % 4. Berechnung des Amplitudenspektrum+++++++++++++++++++++++++++++++
15       dataAbsFFT      = abs(dataComplexFFT);
16       %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
17       % 5.Normierung des Spektrum und Beschränkung auf das Einseiten-Spektrum
18       dataAbsFFT      = 2*dataAbsFFT(1:end/2)/length(data); % Berechnung des
19       % 6. Einseiten-Spektrums
20       dataAbsFFT(1)   = dataAbsFFT(1)/2; % Halbierung des ersten Bins, weil
21       % die Spektrallinien bei 1 und NFFT/2 zusammen fallen
22       % Berechnung der Effektiv-Werte+++++++++++++++++++++++++++++++++++++++
23       dataRMSFFT      = dataAbsFFT/sqrt(2); % Berechnung des Effetivwertes
24       % ==> Crest-Faktor für Sinus-Schwingungen 1/sqrt(2) ~ 0,707
25       %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

In der getsensitivity-Funktion wird der Übertragungsfaktor unter Verwendung eines akustischen Kalibrators bestimmt. Die Kalibrierung erfolgt durch die Auswertung der jeweils aktuellen Signalblöcke. Je nach Abtastrate variiert die Länge eines Blocks zwischen 1024 Samples und 16384 Samples pro Block. Für das Beispiel einer Abtastrate von 48 kHz wird die getsensitivity-Funktion nach jeweils 8192 Samples bzw. nach etwa 171 ms aufgerufen. Vom aktuellen Signalblock wird der Effektivwert berechnet. Dieser wird wiederum in einer Historie abgespeichert. Dieser Vektor dient im späteren Verlauf dazu Pegelschwankungen des Kalibriertons feststellen zu können. Die Kalibrierung ist nur erfolgreich, wenn der Kalibrierton über mindestens 5 Sekunden eine maximale Abweichung von $\pm\ 0{,}5\ dB$ besitzt. Wenn diese Bedingung erfüllt ist wird der Übertragungsfaktor des aktuellen Signalsblocks abgespeichert und die Kalibrierung beendet.
Der Benutzer kann die Übernahme des neuen Übertragungsfaktors auch verweigern, das ist sinnvoll, wenn die Pegeländerungen durch den neuen Übertragungsfaktor größer als ein paar dB sind. In diesem Fall kann ein Fehler oder Schaden in der Messkette vorliegen.

Sobald eine das Mikrofon auf einen neuen Mikrofonübertragungsfaktor kalibriert wurde kann die Kalibrierung auf zwei verschiedene Weisen überprüft werden. Die erste Möglichkeit besteht in dem erneutem Aufruf der Kalibrierungsroutine. Mit dem neuen Übertragungsfaktor sollte der gemessene Pegel relativ schnell den erwartenden Kalibrierpegel anzeigen (z.B. 94 dB) anzeigen. Wenn der Pegel über Fünf Sekunden konstant ist. wird der eine neuer Übertragungsfaktor und die Pegeldifferenz zu gegenüber der vorherigen Empfindlichkeit angezeigt.

3.2.6. STROMVERSORGUNG

Die Stromversorgung stellt einen der kritischen Punkte des Datenerfassungssystems dar. Es müssen insgesamt 16 ICP-Mikrofone, 4 USB-Messmodule, und einen Host-PC mit Strom versorgt werden. Die Stromversorgung der Mikrofone ist in der Signalkonditionierung des DT9837B-Module integriert. Der gesamte Leistungsbedarf des System liegt bei ca. $15{,}5\ W$ (siehe Tabelle 3.5).

Der genaue Bedarf richtet sich nach Auslastung des PCs. Je höher dieser ist, desto größer ist der Leistungsbedarf. Der Anforderungskatalog fordert eine durchgehende Betriebszeit von 8-10 Stunden. Die vier USB-Module werden über zwei USB-Buchsen an den Computer angeschlossen. Ein Modul wird direkt mit dem PC verbunden, die übrigen drei Module werden über ein aktives USB-Hub an den Computer angeschlossen. Die Geräte benötigen für einen zuverlässigen Betrieb einen

Abbildung 3.15: Darstellung des Kalibrierfenster einer erfolgreichen Pegelkalibrierung innerhalb der MAMS-Software-Applikation.

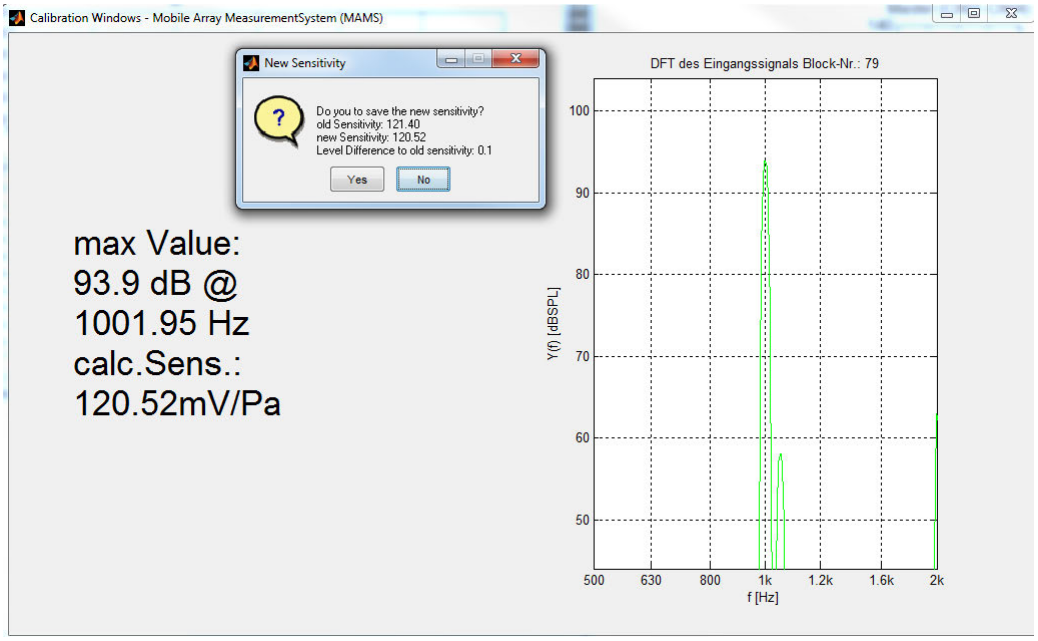


Tabelle 3.5: Leistungsbedarf der Komponenten des MAMS

PC-Host	≤ 15 W
DT9837B-Modul	4 x (0,3 V · 0,425 A)
Gesamt	$\approx 15,5\ W$

Versorgungsstrom von mindesten 425 mA. Da der zur Verfügung stehende Strom eines USB-Ports auf maximal 500 mA begrenzt ist muss an dieser Stelle ein aktiver USB-Hub verwendet werden. Er stellt sicher, dass jedes, an den Hub angeschlossene Gerät, einen Strom von bis zu 500 mA abrufen kann. Der Strom wird mit den internen Akkus des USB-Hub CP-H420MP (4 x AAA Akkus) zur Verfügung gestellt. Die Akkus des USB-Hubs besitzen eine Kapazität von 1300 mAh bei einer Nennspannung von $1,2\ V$. Die Parallelschaltung erhöht die Kapazität des USB-Hubs auf 5200 mAh. Der USB-Hub ist so konzipiert, dass er nur den zusätzlich vom angeschlossenen Gerät benötigten Strom zur Verfügung stellen muss. Der maximale Strom von 500 mA wird auf alle angeschlossenen USB-Geräte eines Ports aufgeteilt. Bei drei DT9837B-Modulen an einem USB-Port werden jedem Gerät ca. 167 mA zur Verfügung gestellt. Für einen ordnungsgemäßen Betrieb ist dieser Strom zu gering. Der USB-Hub stellt jetzt den zusätzlichen Bedarf von $(425 - 167) = 258mA$ an jedem Modul bzw. 774 mA insgesamt zur Verfügung. Die theoretische Betriebsdauer des USB-Hubs beträgt mit diesen Akkus

$$t = \frac{C}{I} = \frac{5200\ mAh}{774mA} = 6,72h \tag{3.10}$$

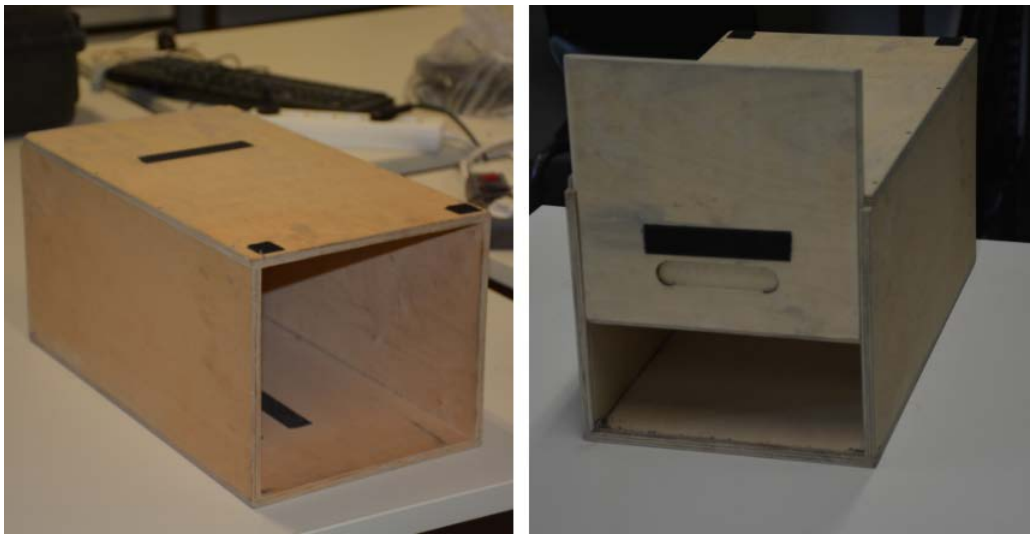
Die Anforderung an die Betriebszeit werden um ca. 1 Stunde unterschritten. Da die Installation der Mikrofone am Array mit mindestens einer Stunde eingeplant wird ist die Betriebszeit des USB-Hub für die DT98337B-Module als akzeptabel einzustufen. Außerdem kann die Laufzeit relativ schnell erhöht werden indem die Akkus ausgetauscht werden. Dadurch werden mit acht Akkus eine Laufzeit von etwa 13 Stundene erreicht.

Die Laufzeit des Laptop beträgt theoretisch etwa 8,5 Stunden[5]. Die Angabe stellt eine messtechnisch ermittelten Messwert unter Laborbedingungen dar. Um die Sicherheit eine langen Betriebsdauer auch bei höherem Leistungsverbrauch des Computers zu ermöglichen wurde eine zusätzliche Powerbank für den Computer vorgesehen. Eine Powerbank mit 23 Ah bei einer Nennspannung von 3,7 V (85,1 Wh) erhöht die Laufzeit auf etwa 14 Stunden ohne an einen Netzstromanschluss angeschlossen werden zu müssen. Als erstes erfolgt die Entladung der externen Powerbank, gefolgt von der externen Akku-Bay und anschließend dem internen Akkus des Computers. Ein größerer Akku bzw. Powerbank kann nicht eingesetzt werden, weil die internationalen Regeln für Flugzeuggepäck nur den Transport eines Akkus mit einer maximalen Kapazität von 120 Wh zu lassen.

3.2.7. GEHÄUSE

Sowohl für den Transportweg mit dem Flugzeug, als auch für den Aufstieg auf den Vulkan wurde ein Gehäuse benötigt in dem die Komponenten des Messsystems aufbewahrt und transportiert werden können. Ein erster Prototyp wurde aus Holzplatten angefertigt. Die Abmessungen wurden so gewählt, dass das Holzgehäuse noch in einen 70l Rucksack gesteckt werden kann und beim Transport mit dem Flugzeug die Abmessungen der Handgepäck-Stücke nicht überschreitet. Die Vorderseite ist zum Verstauen der Messgeräte offen gehalten. Die Rückseite Der Kiste ist mit eine schiebbaren Service-Klappe ausgestattet, damit im Fall eines Fehlers der Zugang zur Rückseite weiterhin möglich ist. Das kann notwendig sein um die Funktion der DAQ-Module über die Status-LEDs oder um Kabel zu überprüfen, ohne alle USB-Module von vorne entnehmen zu müssen. Das Gehäuse ist in der folgenden Abbildung dargestellt.

Abbildung 3.16: Erster Gehäuse-Prototyp des MAMS.



[5]Gesamtenergie der internen und externen Akkus von 127 Wh mit einem typ. Verbrauch von 15 Watt. Der Verbrauch wurde bei eingeschalteten Energie-Manager Profil M̈aximale Lebensdauer des Akkusërmittelt.

4

Evaluation des mobilen Datenerfassungssystems

Mit der Evaluierung des Datenerfassungssystems soll überprüft werden ob die Anforderungen an das System mit den vorhandenen Eigenschaften erfüllt werden.
Die wesentlichen Kriterien aus dem Kapitel 3 lauteten:

- synchrone, Sample-genaue Verarbeitung von 16 Eingangskanälen,
- integrierte IEPE-Signalkonditionierung,
- Frequenzbereich von mindestens 10 Hz bis 20 kHz,
  - optional von 0,1 Hz bis 50 kHz
- linearer Phasengang (konstante Gruppenlaufzeit),
- gutes SNR,
- geringer THD,
- flexible programmierbare Steuerung der Datenerfassung,
- mobiler Batteriebetrieb (8-10 h),
- geringes Gewicht und ein kleines Packmaß.

Die Evaluierung erfolgte in verschiedenen Schritten, da die Anforderungen sehr unterschiedlich waren. Es gab sowohl Labor, als auch Feld-Messungen um die Funktionsfähigkeit des DAQ-Systems zu überprüfen. Im Labor wurden die Funktionsfähigkeit und Bedienung der DAQ-Software-Applikations untersucht. Zusätzlich wurden elektrische Messung zu den Eigenschaften der USB-Messgeräte durchgeführt. Im Feld fand vor der eigentlichen Messung am Vulkan ein Test zum praktischen Aufbau und

Langzeitstabilität des Messaufbaus statt.
Wie bereits zu Beginn erwähnt wurde sind das Trägersystem der Mikrofone, die Array-Geometrie und das Beamforming nicht Gegenstand dieser Arbeit und werden daher nicht bei der Evaluierung berücksichtigt.
Im Folgenden werden in verschiedenen Abschnitte die Eigenschaften der Messsystem-Komponenten mit den Anforderungen verglichen.Im ersten Abschnitt dieses Kapitels wird der Evaluierung der Software vorgenommen. Da die Bearbeitungszeit keine umfangreiche Quality and Usability-Untersuchung mit Probanden zuließ wird an dieser Stelle im wesentlichen nur die Kalibrierungs-Funktion und die Stabilität der Software-Applikation untersucht. Der Bedienungskomfort kann nur aus eigener Perspektive abgeschätzt, aufgrund der fehlenden Test mit Probanden jedoch nicht beurteilt werden.

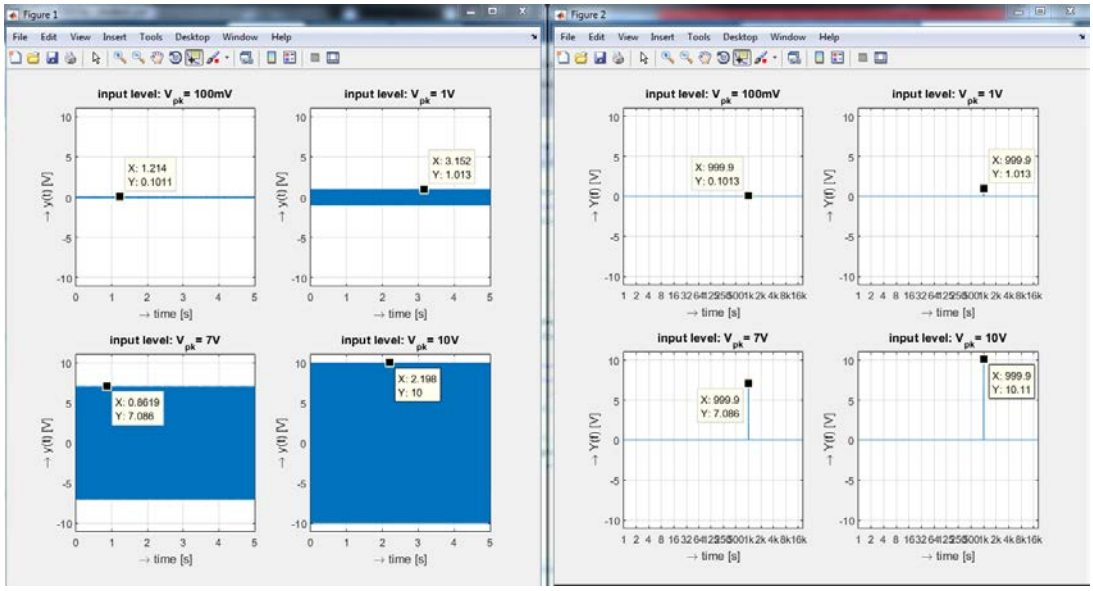
4.1. EVALUATION DER SOFTWARE

4.1.1. KALIBRIERUNG

Der FFT-Analysator der für die Kalibrierung verwendet wird, wird anhand eine elektrischen Testmessung überprüft. Es werden insgesamt fünf Sinussignale mit einer Frequenz von 1 kHz und einer Amplitude zwischen $0,01\ V$ bis $10\ V$ erzeugt und mit einem DT9837B-Modul gemessen. Wenn der Analysator korrekt funktioniert wird im Bereich von $1kHz$ ein Signal mit der identischen Amplitude bzw. Effektivwert erwartet.
Die Messung wurde mithilfe eines dScope Series III Audio Analyzer durchgeführt. Mit dem dScope wurde das Sinussignal an Output A erzeugt und anschließend mit den Eingangskanal des Mastermoduls verbunden. Als Messkabel wurde ein ca. 1 m langes einfach geschirmtes RG58-Kabel verwendet.

Das Ergebnis der Untersuchung wird in der Abbildung 4.1 dargestellt. Die Messsignale sind jeweils fünf Sekunden lang und wurden mit dem Flat-Top-fenster gewichtet. Die Anzahl der FFT-Punkte beträgt $N = 2^18 = 26.2144$. Das führt bei einer Abtastrate von 48 kHz zu einer Auflösung von $\Delta f = 0,92\ Hz/Bin$.

Abbildung 4.1: Darstellung der Zeitsignals und des Spektrum eines Referenzton mit unterschiedlichen Amplituden



Die Abweichungen liegen bis 7 V in einem zulässigen Bereich. Unterhalb einer Eingangsamplitude von 7 V beträgt die Abweichung der maximalen Amplituden ca. $0,2\ mV$ (siehe 4.1). Es ist

auffällig, dass bei der Eingangsamplitude von 10 V die Abweichung der maximalen Amplitude von Spektrum und Zeitsignal relativ groß wird. Das liegt unter Umständen in der Übersteuerung des A/D-Wandler durch diese hohe Amplitude. Daraufhin wird die Energie der harmonische Verzerrungen auch auf die Signalfrequenz.

Tabelle 4.1: Messergebnisse des FFT-Analysator Test für die Kalibrierung.

maximale Amplitude (Zeitsignal) V	maximale Amplitude (Spektrum) V	Differenz V
0,1016	0,1013	0,0002
1,0135	1,0134	0,0001
7,0858	7,0858	0,0000
10,000	10,1082	-0,1082

So ein hohe Pegel wird jedoch auch nicht für eine Pegelkalibrierung verwendet. Mit einer typischen Empfindlichkeit von 50 mV/Pa erzeugt ein Mikrofon an seinem Ausgang ein Signal mit einer Amplitude von etwa 50 mV. Eine Übersteuerung ist nicht zu erwarten. Die FFT-Analysator funktioniert somit erwartungsgemäß in einer angemessenen Qualität.

4.1.2. LANGZEITSTABILITÄT UND BEDIENKOMFORT

Die MAMS-GUI kann nachdem sie einmal gestartet wurde vollständig mit der Tastatur bedient werden. Die Messungen werden mit einem Druck auf die Leertaste gestartet und gestoppt. Sollen bestimmte Ereignisse protokolliert werden, kann der Benutzer mit den anderen Tasten den Zeitpunkt im Eventlog aufnehmen. Die Funktionen im Menü (z.B. Kalibrierung oder Ansicht der Mikrofon-Daten) können per Maus-Klick oder über Short-Cuts erreicht werden.
Aus eigener Erfahrung hat sich die Bedienung der Messung über die Leertaste als sehr komfortable erwiesen. Besonders in rauen Messumgebungen und heller Sonneneinstrahlung ist es mit einem kurzen Blick möglich zu überprüfen ob nach einem Druck auf die Leertaste die Messung gestartet wurde. Dasselbe trifft auf die Kalibrierung zu. Mit den Shortcut STRG+C wird das Menü zur Auswahl des zu kalibrierenden Mikrofons geöffnet. Alle weiteren Entscheidungen können ebenfalls mit der Tastatur getroffen werden. Gerade während einer Feldmessung war die Bedienung mit der Tastatur zuverlässig. sDie Maus wird nur noch bei der Konfiguration des Mikrofon-Setups benötigt, um die Seriennummer und damit die entsprechenden Mikrofone auszuwählen.
Insgesamt bewerteten alle bisherigen Benutzer des MAMS die Bedienung der GUIs als komfortabel. Hier wird der Farbwechsel von grün und rot beginnender oder beendeter Messung hervorgehoben.Jedoch handelt es sich dabei um einen kleinen Personenkreis von weniger als 5 Personen. Deshalb besitzen diese Aussagen eine eingeschränkte Aussagekraft.
Bezüglich der Langzeitstabilität können noch keine eindeutigen Aussagen getroffen werden. Im Rahmen der Entwicklung und Evaluierung war das MAMS mehr als 100 Messstunden in Betrieb. Mit seltenen Ausnahmen gab es eine Fehlermeldung und die DAQ-Module konnten nicht gestartet werden. Nach einem kompletten Neustart von MATLAB war das Problem behoben. Dieser Fehler konnte nicht zuverlässig reproduziert werden. Auch ließen sich keine eindeutige Einflussfaktoren, als der Fehler passiert feststellen.

4.2. EVALUATION DER HARDWARE

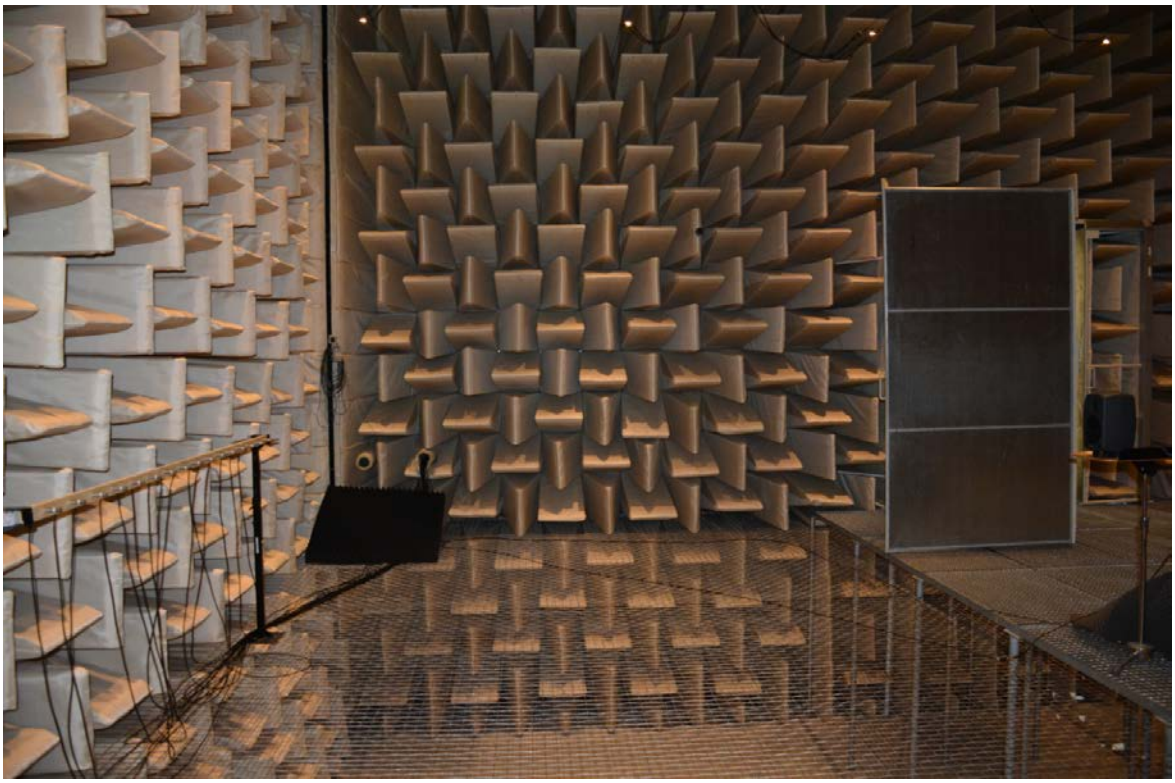
Dies Auswertung der .daq-Dateien erfolgten mit einer aktuellen Version MATLAB R2015a (8.5.0.197613). Diese Version enthält in der Signal Processing Toolbox Algorithmen zur Auswertung von Harmonischen Verzerrungsprodukten wie z.B. dem THD. Darüber hinaus besteht die Möglichkeit Signale bezüglich des SINAD, die SFDR und des SNR mithilfe von den MATLAB-Algorithmen zu berechnen. Es wurden keine eigene Algorithmen verwendet um Fehler auszuschließen und eine Transparenz reproduzierbarer Ergebnisse zu erhalten. Die USB-Messgeräte verfügen über keine Ausgangskanä-

le. Daher war es nicht möglich den Frequenzgang der Geräte zumessen ohne die Herstellergarantie zu verlieren. Deshalb wurde auf diese Messung verzichtet in der Annahme, dass die Herstellerangaben korrekt sind.

4.2.1. LABORMESSUNG

Die nachfolgenden Labormessungen fanden in der Regel Im RAR der TU Berlin statt. Das vorgestellte typische MAMS-Konfiguration wurde im RAR aufgebaut. An jedes der vier DT9837B-Module wurden vier Mikrofone angeschlossen. Die Mikrofone wurden als Linien-Array gegenüber einen Lautsprecher angeordnet.

Abbildung 4.2: Messaufbau zur Bestimmung von Synchronisationsabweichungen im RAR der TU Berlin (links, Mikrofonarray; rechts, Lautsprecher).



Die Module wurden jeweils mit einem USB 2.0 Kabel mit dem Host-PC verbunden. Damit die Datenerfassung der einzelnen Module synchron verläuft wurden die RJ45-Buchse der Module mit kurzen Patchkabel an das EP-Panel angeschlossen mit dem Panel wird das Sync-Signal vom Master-Modul an die angeschlossenen Slaves weiter durchgeschleift.

SYNCHRONISATION DER SIGNALE

Mit den zur Verfügung stehenden Mitteln konnte keine Asynchronizität zwischen den einzelnen Modulen festgestellt werden. Da nicht direkt an der Hardware gemessen werden konnte wurde innerhalb von MATLAB die Untersuchung vorgenommen. Für jedes Modul erzeugt die MAMS-Applikation eine gleichnamige .daq-Datei. Um die Daten auszulesen muss die Datei mit der Funktion daqread() ausgelesen werden.

```
1 [data, time, abstime, events, daqinfo]  = daqread('test.daq');
```

Um die Synchronizität zu überprüfen muss als erste die Vairable abstime überprüft werden. Diese gibt den Zeitpunkt an, wann das Modul mit der Datenerfassung begonnen hat. Dadurch das als

erstes die Slaves in einer Wartehaltung gestartet werden und anschließend über den Start des Master mit der Datenerfassung beginnen müssen alle Module die selbe abstime besitzen. Als nächstes wird die Differenz der Zeitvektoren der Slave-Module zum Master-Modul berechnet. Diese Differenz beschreibt die zeitlichen Abweichung der Samples der Slave-Module zum Master-Modul. Aufgrund der Zwischenspeicherstruktur mit dem Buffer im DAQ-Modul und innerhalb der Matlab-Umgebung sind die Samples die mit der MAMS-Applikation berechnet wurden vollkommen synchron. Auf Seite der Hardware konnte villeicht eine Verzögerung erfolgen, die innerhalb der Software bereits kompensiert wurde.

FFT-ANALYSE DER SIGNALE

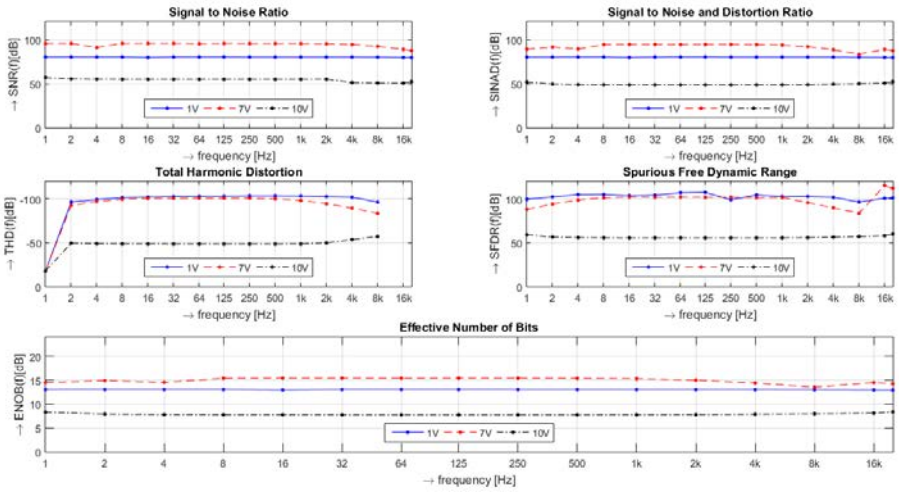
Zur Evaluierung der Eignung der Datenerfassung für die bevorstehende Messaufgabe am Vulkan werden vier Parameter gemessen: das SNR, die SINAD, die THD und der SFDR. Das SNR stellt ein klassischen Parameter dar, welcher das logarithmiertes Verhältnis der Signalleistung zur Rauschleitung dar. Ein besonders gutes SNR besitzt eine großen Wert. Dasselbe gilt für das SINAD, und dde SFDR. Für den Klirrfaktor (THD) gilt es einen möglichst niedrigen Wert zu erzielen. Das THD gibt das Verhältnis der Amplituden harmonischen Verzerrungen zur Eingangsamplitude des Eingangssignal an. Je niedriger das THD ist, desto weniger harmonische Verzerrung werden am Ausgangs des A/D-Wandlers gemessen.
Die Überprüfung der elektrischen Eigenschaften erfolgt auf zwei Wegen. Als erstes werden SNR, THD, SINAD und SFDR in Abhängigkeit der Signalfrequenz im Bereich von 1 Hz bis 20 kHz für drei Signalpegel (1 V, 7 V und 10 V) dargestellt. Nach Auswertung dieser Daten werden die selben Parameter in Abhängigkeit der Eingangspegel mit einer Frequenz von jeweils 1 kHz dargestellt.
Als Messsetup wurde das typische MAMS-Setup mit vier DT9837B Modulen eingesetzt (siehe Abbildung 4.3).

Abbildung 4.3: Vollständig aufgebautes MAMS mit neuem Gehäuse (Pelicase, rechts)

Die Eingangssignale wurden hier nicht mit den Mikrofonen gemessen und analysiert sondern

elektrisch mit einem DScope-Messsystem erzeugt und direkt mit den Eingänge der DAQ-Module verbunden, um den Einfluss der Mikrofone an dieser Stelle vernachlässigen zu können
Die Ergebnisse der frequenzabhängigen Messung werden in der Abbildung 4.4 dargestellt.

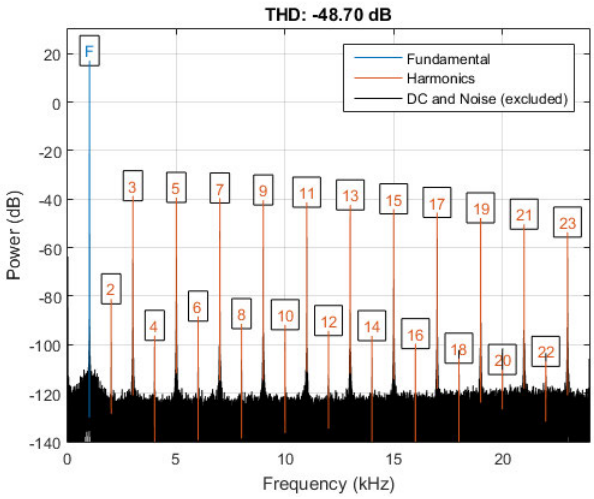
Abbildung 4.4: Gemessene Werte für das SNR, SINAD, THD, SFDR und ENOB in Abhängigkeit der Signalfrequenz gemittelt über 4 Kanäle mit je vier Messungen mit einer Dauer von jeweils 5 Sekunden(Hanning-Fensterung)



Die USB-Messgeräte wurden bei jeder Messung mit einem Eingangsbereich von ± 10 V betrieben. Durch den Messbereich von -10 V bis + 10 V lassen sich für die Signale mit einem Eingangspegel von -20 dBFS(1V), - 3 dBFS(7V) und 0 dBFS zwei Erwartungen ableiten, die in der Abbildung auch wiedergegeben werden. Als erstes wird erwartet, das die alle Werte mit zunehmenden Eingangspegel besser werden. Zweites wird ab einem bestimmten Pegel erwartet, dass die Werte rapide schlechter werden. In der dargestellt Abbildung wird der Frequenzverlauf von fünf Parameter in einem Frequenzbereich von 1 Hz bis 20 kHz dargestellt.
Das SNR zeigt das schlechteste Verhalten bei einem Eingangspegel von 10 V. Hier liegt das durchschnittliche Verhältnis vom SNR bei etwa 50 dB. Die beste Performance zeigt sich bei einem Eingangspegel von 7 V. Das entspricht dem maximalen Ausgangspegel der PCB 387B02 Mikrofonen. Mit sinkenden Pegel wird das Signal zu Rausch Verhältnis ebenfalls geringer. Das schlechtere SNR bei hohen und niedrigen Pegel hat zwei Ursachen. Das SNR berücksichtigt per Definition nur die Rauschleistung ohne harmonische Verzerrungsprodukte oder nicht gefilterten Gleichspannungsanteile. Bei hohen Pegel übersteuert der A/D-Wandler. Es werden die Amplituden, die über dem maximal darstellbaren Eingangsspannungen liegen mit dem höchsten zulässigen Wert kodiert. Dadurch können rechteckähnliche Signal entstehen, weil die Amplitude klippt. Diese Clippen ist mit dem Signal korreliert. Der Quantisierungsfehler ist nun größer als das theoretische Maximum von 0,5 LSB und die Rauschleistung wird auch größer. Die höhere Rauschleistung aufgrund des größer werdenen Quantisierungsfehler führt zu einer Verringerung des SNR.
Bei kleinen Signalpegel verhält es sich anders. Durch den geringen Signalpegel wird der A/D-Wandler nicht voll ausgesteuert. Das Signal mit kleiner Amplitude wird nur mit wenigen gültigen Kodewörtern kodiert. Das entspricht eine Quantisierung mit geringer Auflösung. Es werden häufiger geringe Pegelunterschiede mit dem selben Kodewort kodiert, was wiederum zu einem höheren Quantisierungsrauschen und somit auch geringerem SNR führt. Das Signal mit einem Eingangspegel von -3 dBFS (7V) ist jedoch fast im vollen Aussteuerungsbereich des A/D-Wandler der USB-Messgeräte und zeigt einen guten Wert von etwas 95 dB. Bei allen Eingangspegel zeigt sich ein fast linearers Frequenzverhalten. Die Messung mit dem 7V-Signal weist Abfall zu hohen Frequenzen von 94,39 dB bei 4 kHz zu 87,71 dB bei 20 kHz auf. Die Abweichungen der übrigen Frequenzverläufe sind relativ

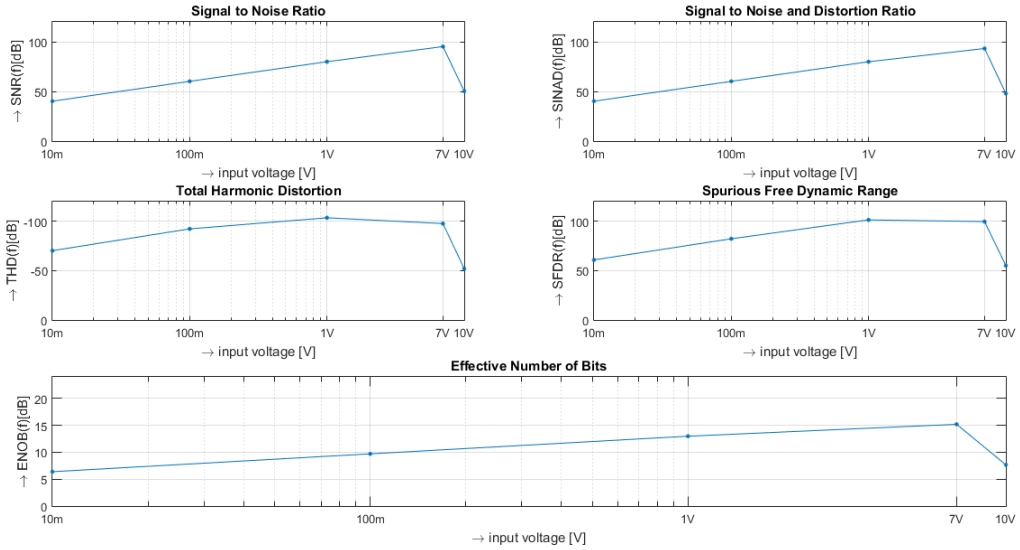
gering.
Der nächste Parameter ist das THD und ist in dieser Arbeit gleichgestellt mit der Bedeutung des klassischen Klirrfaktors. Der Klirrfaktor beschreibt das Verhältnis der Amplitude der Eingangssignalfrequenz zu der Summe der Amplituden der harmonischen Verzerrungen. Für die DT9837B-Module bedeutet das, dass mit einem 1 V- und 7 V-Signal Werte von etwa $-100\ dBc$ erreicht werden. Die Übersteuerung des 10 V Signals führt zu einem $THD = -50\ V$. Dieser Wert ist auf die Verzerrungsprodukte aufgrund der Übersteuerung zurück zuführen. Der Frequenzverlauf zeigt beim THD auch eine Frequenzabhängigkeit mit zunehmender Frequenz auf. Dieser Effekt ist nur bedingt aussagefähig weil mit zunehmender Signalfrequenz weniger Verzerrungsprodukte berücksichtigt werden können,schließlich besitzt der Analysebereich eine Grenzfrequenz von 20 kHz. Deshalb wurde ab einer Frequenz von 8 kHz auf die Darstellung des THD verzichtet.

Abbildung 4.5: Darstellung des FFT-Spektrums zur Ermittlung des THD eines übersteuerten 1 kHz Sinus-Signals gemessen mit einen DT9837B-Modul.



Der vierte Parameter ist der SFDR. Er beschreibt den Dynamikbereich von der Signalamplitude zur ersten Störamplitude im FFT-Spektrum. Das 10 Volt-Signal besitzt erwartungsgemäß die geringste Dynamik aufgrund der starken Verzerrung. Die Dynamik der bei anderen Signale liegen in etwa im selbem Bereich von etwa $80 - 100\ dB$.
In der Messtechnik ist die ENOB, die effektive Anzahl der Bits, ein aussagekräftiger Parameter. Der Parameter wurde aus den gemessenen SINAD-Werten berechnet. Die ENOB beschreiben welche effektive Quantisierungsauflösung der A/D-Wandler besitzt. Durch Verzerrungsprodukte und zusätzliches Rauschen (thermisch, elektrisch) wird der theoretische Wert von 24 Bit nicht erreicht. Im Rahmen dieser Arbeit wurde eine ENOB etwa 7 - 16 Bit ermittelt. Das entspricht einem Verlust von 8 bis 17 Bit zum theoretischem Ideal.
In der Abbildung 4.6 werden die Parameter in Abhängigkeit des Eingangssignalpegel dargestellt. Die Tendenz aus der vorherigen Betrachtung setzt sich hier fort. Mit zunehmender Signalamplitude verbessern sich die jeweiligen Parameter der gemessenen Signal. Ab einer gewissen Eingangsamplitude werden die Werte sofort schlechter. Ab diesen Punkt übersteuert das Signal. Durch die Übersteuerung wird dem Eingangssignal zusätzliches Rauschen und harmonische Verzerrungsprodukte hinzugefügt. Der Messbereich des Prüfling lag bei den Messungen im Bereich von $-10\ V$ bis $+10\ V$ eingestellt. Aus diesem Grund sind die Werte bei einer Eingangsamplitude von 10 mV sehr niedrig. Die geringe Amplitude führt zu einer Quantisierung nahe (45 dB Abstand zum Rauschteppich) des Rauschpegel des Messsystems. Hingegen ist der Abstand beim 7 Volt Signal mit am größten.

Abbildung 4.6: Gemessene Werte für das SNR, SINAD, THD, SFDR und ENOB in Abhängigkeit der Eingangsamplitude gemittelt über 16 Kanäle mit je vier Messungen mit einer Dauer von jeweils 5 Sekunden(Hanning-Fensterung).



ZUSAMMENFASSUNG

Die Ergebnisse zeigen das die DT9837B-Module in ihren Eigenschaften (SNR, SINAD, THS, SFDR) einen relative linearen Frequenzverlauf von 2 Hz bis 4 kHz besitzen. Dasselbe gilt für die effektive Anzahl der Bits. Mit dem vorgestellten Messaufbau konnte jedoch keine Messung mit eine sehr hohen Quantisierungsauflösung reproduziert werden. Die maximale Auflösung lag im Bereich von 15 Bit mit einer Eingangsamplitude von 7 V. Die DAQ-Module reagierten in dem Test empfindlich auf Unter- und Übersteuerung des A/D-Wandlers. Das zeigten auch die Untersuchungen der Module in Abhängigkeit der Eingangsamplitude. Hier wiesen die untersuchten Parameter ihr lokales Maximum bei eine Amplitude von 7 Volt auf. Der DScope-Signalgenerator und das durch die Hardware zusätzlich hinzugefügte Rauschen führten zu Amplitude die wahrscheinlich geringfügig oberhalb der zulässigen Fullscale-Amplitude von 10 Volt lagen. Dadurch wurden starke harmonische Verzerrungsprodukte erzeugt, welche zu einer relevanten Verringerung der Messwerte führte.
Werden die Anforderung jetzt mit den vorliegenden Kenntnisse verglichen lassen sich folgende Schlüsse ziehen.

- Es ist eine integrierte IEPE kompatible Signalkonditionierung vorhanden. Die Konstantstromquelle stellt 4 mA bei einer Versorgungsspannung von 18 V zur Verfügung.

- Bei einem typischen Ausgangsbereich der Mikrofone von $\pm 7\ V$ erreicht das THD einen mittleren Wert von etwa -98 dBc. Dieser Wert liegt in etwa im Bereich der Herstellerangaben [9]. Bei der Übersteuerung des A/D-Wandlers führen starke Verzerrungsprodukte zu einem niedrigen THD.

- Das SNR liegt laut Herstellerangaben bei SNR = 106 dB [9]. Dieser Wert konnte messtechnisch nicht überprüft werden. Das höchste Signal-/Rauschverhältnis wurde mit einer Amplitude von 7 V zu SNR = 95 dB bestimmt. Der Parameter SINAD beschreibt das Verhältnis der Leistung des Eingangssignal zu der Summe der Leistung der Harmonische Verzerrungen und des Rauschteppichs. Das höchste SINAD wurde zu ca. 93 dB bestimmt. Eine Herstellerangabe

zu diesen Parameter gab es nicht. Das SFDR liegt unter guten Messbedingungen bei etwas 100 dBc (Input = 1 V und 7 V) und entspricht in etwa den Herstellerangaben.

- Neben den elektrischen Eigenschaften lässt sich feststellen, dass durch die Kombination der Data Translation Open Layer Struktur und der Verwendung von MATLAB und der DAQ-Toolbox eine flexible Programmierumgebung zur Verfügung stellt. Mit den programmierbaren Benutzeroberflächen können alle gewünschten Funktionen auch grafisch umgesetzt werden ohne ein zu vertieftes Programmierwissen zu benötigen.

- Der Betrieb des MAMS ist ohne größerer Probleme für mehr als 10 h möglich. Durch den aktiven batteriebetriebenen USB-Hub können drei USB-Messmodule an einem USB-Port mit Strom versorgt werden. Nach 6-7 Stunden müssen die Akkus des USB-Hubs getauscht werden. Der Host-PC wird über eine internes Akku, einer zusätzlichen Akku Bay und einer Powerbank mit Strom versorgt. Die Kapazität der Akkus und der geringe Stromverbrauch ermöglichen einen etwa 14 stündige Betrieb.

- Auch die Anforderungen an das Gewicht werden erfüllt. Jedes der Module wiegt ca 500 g. In der Summe wiegen sie 2 kg. Das Gewicht des Computer und der Peripherie berücksichtigend kommt das MAMS inklusive des Holz-Protoyp-Gehäuses auf ein Gesamtgewicht von etwas 5,5 kg.
- Das die DT9837B Module synchron, Sample-genaue arbeiten konnte im Rahmen der Messungen auch bestätigt werden. Die Strukutr mit dem Zwischenspeicher sorgt für eine softwareseitig absolut synchrone Verarbeitung. Am Eingang kann die Erfassung asynchron sein, das wird jedoch durch die Zwischenspeicherstruktur abgefangen.

- Im Rahmen dieser Arbeit konnte kein genauer Frequenzbereich bzw. der lineare Phasengang (resultiert in einer konstanten Gruppenlaufzeit) nachgewiesen werden, weil kein Signalausgang zu Verfügung steht. Da es sich um eine zeitkritische Entwicklung handelte wurde auf das Öffnen des Gehäuses verzichtet. Der Hersteller hätte die Garantie entzogen und das wurde nicht beabsichtigt, weil das Messsystem sich noch in der Entwicklung befand und etwaige Fehler seitens des Hersteller nicht hätten reklamiert werden können.

4.2.2. FELDMESSUNGEN

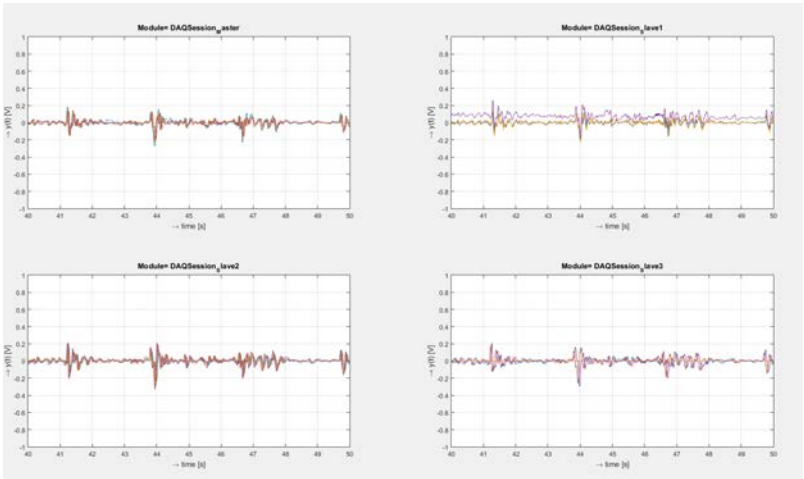
Neben den Labormessungen fand vor der eigentlichen Messkampagne in Italien eine Messung auf dem Teufelsberg in Berlin statt. Die Messungen auf dem Teufelsberg fand Anfang Mai statt und hatte verschiedene Ziele.
Für die Datenerfassung war es wichtig zu sehen ob die Remote-Desktop Verbindung vom Host-PC zu einem Remote-PC funktioniert und stabil ist. Auf dem Teufelsberg wurde sowohl eine drahtgebunde (LAN), als auch eine drahtlose (WLAN) Verbindung zwischen beiden Computern hergestellt und überprüft. Die LAN-Verbindung mit einem 50 m langen Kabel zeigt keine Probleme. Im Abstand von etwa 15 m - 20 m wurde die Verbindungqualität der WLAN Verbindung schlechter und die Remote-Desktopverbindung brach ab. Der Messprozess wurde automatische fortgeführt konnte zu diesem Zeitpunkt aber nicht mehr geändert werden (ohne aktive Remote Vebindung). Im wesentlichen konnte kurz vor der eigentlichen Messung das Konzept der Datenerfassungssystems überprüft werden und noch kurzfristig Probleme im Feld bestimmt werden. Bezüglich der Datenerfassung war der Testaufbau erfolgreich. Die Spannungsversorgung der DT9837-Module lief mit den akkubetriebenen USB-Hub sehr stabil. Die Powerbanks versorgten die beiden Computer mit der notwendigen

Spannung. Es wurden einige Änderungswünsche in der Bedienung der Oberfläche notiert, die bis zur Messung im Mai in 2014 geändert werden sollten.
Im Mai fand dann die Forschungsreise zum Stromboli-Vulkan auf der gleichnamigen Insel in Italien statt. Vom 15. Mai bis zum 21 Mai war Zeit Messungen durchzuführen. Leider war die Witterungsbedingungen am Anfang so schlecht, das erst am vierten Tag Messungen mit dem MAMS durchgeführt werden konnten. Grund für die Verzögerung war der schwierige Aufbau des Mikrofonträgersystems an einem Gipfelhang gegenüber des Vulkans (siehe Abbildung 4.7).

Abbildung 4.7: Messort des Mirkofonarray am einem Gipfel gegenüber des Vulkans Stromboli.

Das Konzept des MAMS hat während der Messungen auf dem Stromboli funktioniert. Die DAQ-Module wurden mit dem Host-Pc und einer Powerbank auf dem Berghang in der Mitte des Kreisarrays installiert. Von dort aus wurden die Messkabel von den Modulen zu den Mikrofonen gelegt. Durch die gefährliche Lage der Mirkrofonpositionen konnte nur eine Person die Installation des Arrays vornehmen. Dazu war es notwendig diese Person zu an einem Seil zu sichern. Mit einem 50 m langen LAN-Kabel wurde die Verbindung vom Host-Pc zum Remote-Desktop oberhalb des Gipfels hergestellt.

Abbildung 4.8: Mit dem MAMS erfasste akustische Signale des Stromboli-Vulkan vom 18.05.2014



5

Fazit und Ausblick

Die Aufgabe dieser Arbeit war es ein Datenerfassungssystems zu entwickeln welches leicht, kompakt, sehr leistungsfähig und dabei stromsparend ist. Diese Arbeit hat sich diesem komplexen Thema genähert in dem die Grundlagen jeder Einzelkomponente untersucht wurden. Aus den verschiedenen Anforderungen die an das System gestellt wurden, wurde nach einiger Recherchezeit klar, dass nicht alle Systeme für diese Aufgabe geeignet sind.
Es zeigte sich, dass modulare USB-Messgeräte einen guten Kompromiss zwischen der Leistungsfähigkeit bzgl. des eigenen Rauschverhaltens und einem stromsparenden und langen Betrieb an einem Messort darstellen. Auch für USB-Messgeräte gibt es eine Vielzahl von Anbietern deren Produkte sich teilweise erheblich in den technischen Daten und auch im Preis unterscheiden. Bedingt durch die Vorauswahl der ICP-Mikrofone wurde die Entscheidung für die Geräteserie DT 9837B der Firma Data Translation getroffen. Diese Geräte werden vom Hersteller mit MATLAB-kompatiblen Treibern versehen. Aus diesem Grund war es möglich die Software für die DAQ-Applikation vollständig in Eigenleistung mit einem zuverlässigen FFT-Analysator zur Pegelkalibrierung von Mikrofonen zu implementieren.
Sowohl die Hardware, als auch die Software wurden zuerst unter Laboruntersuchungen auf ihre korrekte Funktionsfähigkeit überprüft. Bei der Software war es sehr wichtig, die Kalibrierung vernünftig zu testen. Ein Fehler in dieser Funktion würde einen falschen Kalibrierungsfaktor berechnen und die dargestellten Pegel wären falsch. Es müsste eine neue Kalibrierung stattfinden um absolute Schalldruckpegelmessungen machen zu können.
Das Datenerfassungssystem wurde für den Einsatz unter schwierigen Umgebungsbedienungen ohne lokaler Stromversorung entworfen. Ein erster Test wurde auf dem Teufelsberg in Berlin vor der Expedition nach Italien durchgeführt. Hier sind keine schwerwiegenden Fehler aufgetreten. Es wurden lediglich ein paar Bedien-Elemente der GUI angepasst um auf Stromboli eingesetzt zu werden. Die zweite Feldmessung fand dann auf Stromboli statt. Weil gerade zu Beginn der Untersuchungen auf dem Vulkan eine schwierige Wetterlage herrschte konnte das MAMS nur an einem Tag für mehrere Stunden als volles Array eingesetzte werden. In dieser Zeit wurden die Messaufgaben jedoch zuverlässig erfüllt.
Die Arbeit an einem Datenerfassungssystem ist eine große Herausforderung. In den 1,5 Monaten der Entwicklungsphase war es nicht möglich alle Eigenschaften und Fähigkeiten der DT9837B-Module zu erfassen und umzusetzen. Daher ergibt sich der Wunsch nach mehreren Umsetzungen von Maßnahmen zur Verbesserung des Messsystems. Einer der wichtigsten Punkte ist die variable Anpassung des Aussteuerungsbereiches. In der jetzigen Version arbeitet das MAMS lediglich im Bereich von -10 V bis + 10 V. Wenn die Signale kleiner als ± 1 V ist empfiehlt es sich den Messbereich auf diesen einzustellen um den A/D-Wandler möglichst voll auszusteuern. Es wäre sicherlich darstellbar über die InputOverrange Condition der Module eine adaptives Verhalten hinzuzufügen.

Ohne diese Änderung zeigen die DT9837B-Module ein befriedigendes Verhalten bzgl. SNR, THD etc. Das SNR liegt im Mittel bei etwa 95 dB. Das liegt ca. 48 dB unterhalb des theoretischen Maximus für einen 24-Bit A/D-Wandler. Die Differenz ist unter anderem auf das zusätzliche Rauschen der IEPE-Konditionierung zurückzuführen, Der Wert steigt sicherlich, wenn die maximale zu untersuchende Amplitude von -0,5 dBFS auch untersucht worden wäre. Dieser Wert wurde bei der Messreihe nicht berücksichtigt.
Neben diesen wesentlichen Änderungen gibt es eine Reihe von Code-Optimierungen zur Optimierung auf weniger Redundanz und mehr Stabilität (z.B. mehr Kurzanweisung zur mehrfachen Kanalkonfiguration verwenden).
Es gab auch bereits Vorschläge zur Optimierung der GUI um einen übersichtlichen Setup-Konfigurator, um einen neuen Messaufbau noch besser zu verwalten. Im Zuge der Vereinfachung der Bedienung soll auch das Laden und Speichern der Objekte ermöglicht werden.
Eine weitere geplante Nutzungserweiterung ist für die Kalibrierung vorgesehen. Anstelle eines fest definierten Pegels von 94 $dBSPL$ sollen weitere Pegel zulässig sein. Das hat den Vorteil, das bei einer Umgebung mit hohem Fremdgeräuschpegel der Kalibrierton z.B. auf 114 dBSPL gestellt erden kann um dennoch eine erfolgreiche Kalibrierung durchführen zu können.
Sollten vergleichbare Funktionen etablierte Schallepegelmessysteme gewünscht werden, wären in einem nächsten Schritt verschiedene Zeit- und Frequenzbewertungen zu implementieren.
Trotz der geplanten Erweiterungen und der Einschränkungen in Bezug auf das Eigenrauschen hat sich das MAMS bereits bewährt. Es wurde seit Juni 2014 in Labormessungen in Waakirchen und der TU Berlin, sowie weiteren Feldmessungen auf dem Stromboli und dem Etna eingesetzt. Dieses Jahr findet eine erneute Expedition auf Stromboli statt. Bei dieser Reise wird das MAMS jedoch aufgeteilt und an drei Orten zeitgleich gemessen.

Literaturverzeichnis

[1] Park, J. und S. Mackay (2003): *Practical data acquisition for instrumentation and control systems*. Amsterdam and London: Elsevier.

[2] Möser, M. (2012): *Technische Akustik*. 9., aktualisierte Auflage. Berlin and Heidelberg: Springer Vieweg.

[3] Weinzierl, S. (2014): „Material zur Vorlesung zu Audiotechnik I im Sommersemester 2014."

[4] Brandt, A. (2011): *Noise and Vibration Analysis: Signal Analysis and Experimental Procedures*. West Sussex: John Wiley & Sons.

[5] Metra Mess- und Frequenztechnik (2015): „IEPE-Standard." URL http://www.mmf.de/iepe-standard.htm.

[6] Aziz, P. M.; H. v. Sorensen und van der Spiegel, J. (1996): „An Overview of Sigma-Delta Converters: How a 1-bit ADC achieves more than 16-bit Resolution." In: *IEEE Signal Processing Magazine*, 13(1) S. 61–84.

[7] MathWorks (2013): „Data Acquisition Toolbox - User's Guide - R2013b."

[8] Meyer, M. (2011): *Signalverarbeitung: Analoge und digitale Signale, Systeme und Filter*. 6., korrigierte und verbesserte Auflage.

[9] Data Translation (2014): „DT9837 Series User's Manual."

[10] Arason, P.; G. N. Petersen und H. Bjornsson (2011): „Observation of the altitude of the volcanic plume during the eruption of Eyjafjallajökull, April-May 2010." In: *Earth System Data Disscussion*, 4(1) S. 1–25.

[11] Holzapfel, N. (2013): „Die vielfätige Bedrohung durch Vulkanasche." URL https://www.uni-muenchen.de/forschung/news/2013/f-46-13.html#top.

[12] Stampka, K.; J. Sesterhenn; C. Graurock und J. J. Pena Fernandez (2014): „Arraymessung von Ausbrüchen des Vulkan Stromboli." In: DEGA (Hrg.) *Fortschritte der Akustik - DAGA 2014*. Berlin: DEGA, S. 856–857.

[13] Scarlato, P. et al. (2014): „The 2014 Broadband Acquisition and Imaging Operation (BAcIO) at Stromboli Volcano (Italy)." In: *2014 AGU Fall Meeting*.

[14] Woulff, G. und T. R. McGetchin (1976): „Acoustic Noise from Volcanoes: Theory and Experiment." In: *Geophys. J. R. astro. Soc.*, (45) S. 601–616.

[15] Matoza, R. S. et al. (2009): „Infrasonic jet noise from volcanic eruptions." In: *Geophysical Research Letters*, (36).

[16] Fee., D. und R. S. Matoza (2013): „An overview of volcano infrasound: From hawaiin to plinian, local to global." In: *Journal of Volcanology and Geothermal Research*, (249) S. 123–139.

[17] Ripepe, M. und E. Marchetti (2002): „Array tracking of infrasonic sources at Stromboli volcano." In: *Geophysical Research Letters*, 29(22).

[18] Johnson, J. B. und M. Ripepe (2011): „Volcano infrasound: A review." In: *Journal of Volcanology and Geothermal Research*, (206) S. 61–69.

[19] Johnson, J. B. (2007): „On the relation between infrasound, seismicity, and smal pyroclastic explosions at Karymsky Volcano." In: *Journal of Geophysical Research*, (112).

[20] Taddeucci, J. et al. (2014): „High-speed imaging, acoustic features and aeroacoustic computations of jet noise form Strombolian (and Vulcanian) explosions." In: *Geophysical Research Letters*, (41) S. 3096–3102.

[21] Fee, D.; A. Yokoo und J. B. Johnson (2014): „Introduction to an Open Community Infrasound Dataset from the Actively Erupting Sakurajima Volcano, Japan." In: *Seismological Research Letters*, 85(6) S. 1151–1162. doi:\url{10.1785/0220140051}.

[22] Henn, H.; G. R. Sinambari und M. Fallen (2008): *Ingenieurakustik: Physikalische Grundlagen und Anwendungsbeispiele: Mit 319 Abbildungen und 36 Tabellen.* 4., überarbeitete und erweiterte Auflage. Wiesbaden: Vieweg + Teubner.

[23] Tam, C. K. W. (1995): „Supersonic Jet Noise." In: *Annu. Rev. Fluid Mech.*, (27) S. 17–43.

[24] Matoza, R. S.; D. Fee; T. B. Neilsen und Gee, K. L. Ogden, D. E. (2013): „Aeroacoustics of volcanic jets: Acoustic power estimation and jet velocity depedence." In: *Journal of Geophysical Research*, 118(12) S. 6269–6284.

[25] Fagents, S. A.; Gregg, T. K. P. und Lopes, R. M. C. (Hrgs.) (2012): *Modeling Volcanic Processes: The Physics and Mathematics of Volcanism.* Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, Sao Paulo, Delhi, Mexico City: Cambridge University Press.

[26] Schneider, M. (2008): „Mikrofone." In: S. Weinzierl (Hrg.) *Handbuch der Audiotechnik*. Berlin: Springer, S. 313–419.

[27] DIN EN 60268-4(2011): „Elektroakustische Geräte - Teil 4: Mikrofone."

[28] Slavik, K. M. (2008): „Anschlusstechnik, Interfaces, Vernetzung." In: S. Weinzierl (Hrg.) *Handbuch der Audiotechnik*. Berlin: Springer, S. 945–1033.

[29] Weinzierl, S. und A. Lerch (2008/2009): *Kommunikationtechnik II.* Vorlesungsskript, TU Berlin, Berlin.

[30] Kester, W. (2008): „Unterstand SINAD, ENOB, SNR, THD, THD+N, and SFDR so You Don't Get Lost in the Noise Floor."

[31] Cerna, M. und Harvey F. A. (2000): „The Fundamentals of FFT-Based Signal Analysis and Measurement."

[32] Michel, U. und M. Möser (2010): „Akustische Antennen." In: M. Möser (Hrg.) *Messtechnik der Akustik*. Berlin, Heidelberg: Springer-Verlag.

[33] Möser, M. (2009): *Technische Akustik.* 8., aktualisierte Auflage. Dordrecht, Heidelberg, London, New York: Springer.

A

Anhang

A.1. MATLAB-Files

A.1.1. MAMS Software-Applikation

Die folgenden MATLAB-Dateien sind alphabetisch geordnet. Zur Ausführung des Applikation muss die MAMS_GUI_v8.m gestartet werden. Diese wird nur korrekt ausgeführt, wenn mindestens ein DT9837B-Modul an dem ausführendem Computer angeschlossen ist und die entsprechenden Treiber installiert sind.

```
function out = calcSpl(aInput)
%-------------------------------------------------------------------------
% Schnittstellenbeschreibung
%
% Autor:                Rick Plescher, 30.04.2015
%
% Funktionsaufruf:      out = calcSpl(aInput)
%
% Fkt.-beschreibung:    Die Funktion calcSpl.m rechnet die Einheiten des
%                       erfassten Signal von Volt in Pascal um.
%
% Eingabe:
%   aInput:             Struktur mit den Eingabeparamtern
%       .signal:        Amplitudenzeitsignal [Volt]
%       .sensitivity:   Mikrofon-Übertragungsfaktor
%
% Ausgabe:
%   out:                Schalldruckpegel in dB re 20 uPa
%
% sontiges:
%
%-------------------------------------------------------------------------

%% Intialisierung
referencePressure = 2E-5; % Bezugsgröße p0= 2 * 10^-5 Pa
%-------------------------------------------------------------------------
%% Programmcode
soundLevelPressure = 20*log10(abs((...
    aInput.signal/aInput.sensitivity)/referencePressure));
%-------------------------------------------------------------------------
%% Wegschreiben der Ergebnisse
out = soundLevelPressure;
%-------------------------------------------------------------------------
```

```
end
```

```
function calibrateMic(aInputObject)
%-------------------------------------------------------------------------
% Schnittstellenbeschreibung
%
% Autor:                Rick Plescher, 30.04.2015
%
% Funktionsaufruf:      calibrateMic(aFigureHandle)
%
% Fkt.-beschreibung:    Diese Funktion initialisiert ein neues Fenster (zur
%                       Kalibrierung. Ein Auswahl-Menü fragt den Benutzer
%                       nach dem zu kalibrierenden Kanal. Nachfolgend wird
%                       das entsprechende DAQ-Modul konfiguriert. Die
%                       eigentliche Kalibrierung erfolgt über die Funktion
%                       getSensitivity.m (ausgeführt als
%                       SamplesAcquieredFunction)
%
% Eingabe:
%   aInputObject        : typ. das Handle zum Figure der MAMS-GUI
%
% Ausgabe:
%   aInputObject        : Indirekte Ausgabe über die aktualisierte
%   (INDIREKT)            Objekt-Spezifische Datenstruktur von der MAMS-GUI
%
% sontiges:
%   notwendig zum Ausführen ist folgende .m-Datei
%       - getSensitivity.m
%   Außerdem muss eine aktuelle microphoneData.mat Datei im Hautpfad
%   liegen.
%-------------------------------------------------------------------------
%% Einlesen der Eingangsdaten
% Einlesen der Eingabedaten der GUI-Datenmatrix
userData            = get(aInputObject,'UserData');

% Einlesen der Mikrofondaten-----------------------------------------------
readMicInput.file   = fullfile(cd,'microphoneData.mat');
readMicInput.header = false;
microphoneData      = readMicrophoneData(readMicInput);

numActiveChannels   = size(microphoneData,1); % Anzahl der aktiven Kanäle

% Auswahl des zu kalibrierenden Mikrofon-----------------------------------
listString = cell(1,numActiveChannels);
for n=1:numActiveChannels
    listString(n) = {[...
        num2str(n,'%02d'),'-',...
        microphoneData{n,4},' (',...
        microphoneData{n,5},') :',...
        microphoneData{n,7}]};
end
% Channelnum enthält den Index des zu kalibrierenden Mikrofons
channelNum = ...
    listdlg(...
        'fus'           ,1                                          ,...
        'ffs'           ,1                                          ,...
        'ListSize'      ,[320,260]                                  ,...
        'PromptString'  ,'Which Microphone do you want to calibrate?',...
        'SelectionMode' ,'Single'                                   ,...
```

```
        'ListString'    ,listString);
%-----------------------------------------------------------------------
%% Programmcode
% Wenn ein Mikrofon ausgewählt wurde wird ein Kalibrierungsfenster
% aufgerufen und dort mit der Kalibrierung fortgefahren
if isempty(channelNum)==0
    % Benutzer-Abfrage ob die Kalibrierung durchgeführt werden soll
    button  =...
        questdlg(...
            'Do you want to start the calibration process?',...
            'Start Calibration Process',...
            'Yes',...
            'No',...
            'No');

    if strcmp(button,'Yes')
        hCalibPlot          = zeros(1,2); % zwei Plot Handles
        hCalibTitle         = zeros(1,2); % zweit Titel-Handles
        hCalibTxtDisplay    = zeros(1,2); % zwei Display-Handles
        hCalibrationHandle  = figure; % Handle des Kalibrierung-Fenster

        % Hintergrundfarbe einlesen
        btnColor            = get(0,'DefaultUIControlBackgroundColor');

        % Konfiguration des Kalibrierungsfenster---------------------------
        % Color             := Hintergrundfrabe der GUI
        % Colormap          := spezifische Farbe der Figure
        % HandleVisibility  := Zugriffsmöglichkeit auf das Handle
        % IntegerHandle     := Vergabe Modus der Figure-Zahl
        % MenuBar           := Anzeige eines Menüs
        % Name              := Name des Fensters
        % NumberTitles      := Anzeige der Figure-Zahl
        % Pointer           := Zeiger-Darstellung
        % Position          := Angabe der Position und Größe
        % Units             := Einheit des Fensters
        % UserData          := Objekt-spezifische Daten
        % Visible           := Sichtbarkeits des Fensters

        set(hCalibrationHandle,...
            'Units'             , 'pixels'                      ,...
            'Color'             , btnColor                      ,...
            'Colormap'          , []                            ,...
            'HandleVisibility'  , 'on'                          ,...
            'IntegerHandle'     , 'off'                         ,...
            'MenuBar'           , 'None'                        ,...
            'Name'              , ['Calibration Windows - '     ,...
                                   'Mobile Array Measurement'   ,...
                                   'System (MAMS)']             ,...
            'NumberTitle'       , 'off'                         ,...
            'Pointer'           , 'arrow'                       ,...
            'Position'          , [100  , 50    , 1024  , 600]  ,...
            'UserData'          , []                            ,...
            'Visible'           , 'on');
        % Konfiguration des suplot 122:= FFT-Plot--------------------------
        subplot(1,2,2)
        hCalibPlot(2)   = ...
            plot(zeros(userData.blockLengthDAQ/2,1));  % Leerer Plot
        hCalibTitle(2)  = ... % Anzeige des Titel
            title(['DFT des Eingangssignals Block-Nr.: ', num2str(0)]);
        xlabel('f [Hz]')
        ylabel('Y(f) [dBSPL]')
```

```
119         grid on;
120         set(gca,... % Konfiguration der X- und Y-Achse
121             'XLim'      , [500,2000],...
122             'XScale'    , 'log',...
123             'XTick'     , [500  ,630  ,800  ,1e3 ,1250  ,1.6e3 ,2e3],...
124             'XTickLabel', {'500','630','800','1k','1.2k','1.6k','2k'},...
125             'YLim'      , [44,104]);
126         % Konfiguration des suplot 121:= Text-Information------------------
127         subplot(1,2,1)
128         hCalibTitle(1) = title('Calibration Information');
129         set(gca,'Visible','Off');
130
131         hCalibTxtDisplay(1) = text(-0.2,0.5,' ','FontSize',26);
132         % Definition der Kalibrierungs-Datenstruktur-----------------------
133         userData.calibrationWindowStruct.hCalibrationHandle =...
134             hCalibrationHandle;
135         userData.calibrationWindowStruct.plotHandle         = ...
136             hCalibPlot;
137         userData.calibrationWindowStruct.txtDisplayHandle   = ...
138             hCalibTxtDisplay;
139         userData.calibrationWindowStruct.titleHandle        = ...
140             hCalibTitle;
141
142         userData.dateStr                                    = ...
143             datestr(now,30);
144         % Aktualisierung der Objekt-spezifischen Datenstruktur
145         set(aInputObject,'UserData',userData);
146         % Fallabfrage in Abhängigkeit der Anzahl der angeschlossenen
147         % DAQ-Module
148         switch userData.numDaqModules
149             case 1 % 1 angeschlossenes Modul: Master
150                 if (channelNum>=1 && channelNum <=4)
151                     try % Try/catch -Abfrage zum Fehler abfangen
152                         % Einlesen der bisherigen Syncronisierungs-Modus
153                        oldSyncMode     = get(userData.master,'SyncMode');
154                         % Konfiguration des DAQModul für die Kalibrierung
155                        set(userData.master,...
156                            'LogFileName'                   , ...
157                                fullfile(userData.folderStr , ...
158                                'DAQSession_Calibration'    , ...
159                                ['Calibration_Channel_'     , ...
160                                 num2str(channelNum,'%02d') , ...
161                                 '_',userData.dateStr])     , ...
162                           'SamplesAcquiredFcn'            , ...
163                                {@getSensitivity_v2,...
164                                 aInputObject,channelNum,oldSyncMode},...
165                            'SyncMode'                      , 'None');
166 % % DUMMY-VERSION--Konfiguration des DAQModul für die Kalibrierung
167 % %            oldSyncMode = [];
168 % %                        set(userData.master,...
169 % %                            'LogFileName'                   , ...
170 % %                                fullfile(userData.folderStr , ...
171 % %                                'DAQSession_Calibration'    , ...
172 % %                                ['Calibration_Channel_'     , ...
173 % %                                 num2str(channelNum,'%02d') , ...
174 % %                                 '_',userData.dateStr])     , ...
175 % %                           'SamplesAcquiredFcn'            , ...
176 % %                                {@getSensitivity_v2,...
177 % %                                 aInputObject,channelNum,oldSyncMode});
178                         start(userData.master); % Start der Kalibrierung
179
```

```
                catch % Falls die Kofiguration des Moduls und
                      % das Starten der Datenerfassung fehlschlägt
                      % wird eine Fehlermeldung ausgegeben und die
                      % Kalibreirung gestoppt
                    errordlg('calibration error');
                end
            end
        case 2 % 2 angeschlossene Module: Master - Slave1
            % Master-----------------------------------------------------
            if (channelNum>=1 && channelNum <=4)
                try
                    oldSyncMode     = get(userData.master,'SyncMode');

                    set(userData.master,...
                        'LogFileName'                   , ...
                            fullfile(userData.folderStr , ...
                            'DAQSession_Calibration'    , ...
                            ['Calibration_Channel_'     , ...
                             num2str(channelNum,'%02d') , ...
                             '_',userData.dateStr])     , ...
                        'SamplesAcquiredFcn'            , ...
                            {@getSensitivity_v2,...
                             aInputObject,channelNum,oldSyncMode},...
                        'SyncMode'                      , 'None');
                    start(userData.master);
                catch
                    errordlg('calibration error');
                end
            end
            % Slave1-----------------------------------------------------
            if (channelNum>=5 && channelNum <=8)
                try
                    oldSyncMode     = get(userData.slave1,'SyncMode');

                    set(userData.slave1,...
                        'LogFileName'                   , ...
                            fullfile(userData.folderStr , ...
                            'DAQSession_Calibration'    , ...
                            ['Calibration_Channel_'     , ...
                             num2str(channelNum,'%02d') , ...
                             '_',userData.dateStr])     , ...
                        'SamplesAcquiredFcn'            , ...
                            {@getSensitivity_v2,...
                             aInputObject,channelNum,oldSyncMode},...
                        'SyncMode'                      , 'None');
                    start(userData.slave1);
                catch
                    errordlg('calibration error');
                end
            end
        case 3 % 3 DAQ-Module : Master-Slave1-Slave2
            % Master-----------------------------------------------------
            if (channelNum>=1 && channelNum <=4) % Master
                try
                    oldSyncMode     = get(userData.master,'SyncMode');

                    set(userData.master,...
                        'LogFileName'                   , ...
                            fullfile(userData.folderStr , ...
                            'DAQSession_Calibration'    , ...
                            ['Calibration_Channel_'     , ...
```

```
                                num2str(channelNum,'%02d') , ...
                                '_',userData.dateStr])     , ...
                            'SamplesAcquiredFcn'           , ...
                                {@getSensitivity_v2,...
                                 aInputObject,channelNum,oldSyncMode},...
                            'SyncMode'                     , 'None');
                        start(userData.master);
                    catch
                        errordlg('calibration error');
                    end
                end
                % Slave1-----------------------------------------------------
                if (channelNum>=5 && channelNum <=8)
                    try
                        oldSyncMode     = get(userData.slave1,'SyncMode');

                        set(userData.slave1,...
                            'LogFileName'                  , ...
                                fullfile(userData.folderStr , ...
                                'DAQSession_Calibration'    , ...
                                ['Calibration_Channel_'     , ...
                                 num2str(channelNum,'%02d') , ...
                                 '_',userData.dateStr])     , ...
                            'SamplesAcquiredFcn'           , ...
                                {@getSensitivity_v2,...
                                 aInputObject,channelNum,oldSyncMode},...
                            'SyncMode'                     , 'None');
                        start(userData.slave1);
                    catch
                        errordlg('calibration error');
                    end
                end
                % Slave2-----------------------------------------------------
                if (channelNum>=9 && channelNum <=12)
                    try
                        oldSyncMode     = get(userData.slave2,'SyncMode');

                        set(userData.slave2,...
                            'LogFileName'                  , ...
                                fullfile(userData.folderStr , ...
                                'DAQSession_Calibration'    , ...
                                ['Calibration_Channel_'     , ...
                                 num2str(channelNum,'%02d') , ...
                                 '_',userData.dateStr])     , ...
                            'SamplesAcquiredFcn'           , ...
                                {@getSensitivity_v2,...
                                 aInputObject,channelNum,oldSyncMode},...
                            'SyncMode'                     , 'None');
                        start(userData.slave2);
                    catch
                        errordlg('calibration error');
                    end
                end
            case 4
                % Master-----------------------------------------------------
                if (channelNum>=1 && channelNum <=4) % Master
                    try
                        oldSyncMode     = get(userData.master,'SyncMode');

                        set(userData.master,...
                            'LogFileName'                  , ...
```

```
                                fullfile(userData.folderStr , ...
                                'DAQSession_Calibration'    , ...
                                ['Calibration_Channel_'     , ...
                                 num2str(channelNum,'%02d') , ...
                                 '_',userData.dateStr])     , ...
                            'SamplesAcquiredFcn'            , ...
                                {@getSensitivity_v2,...
                                 aInputObject,channelNum,oldSyncMode},...
                            'SyncMode'                      , 'None');
                        start(userData.master);
                    catch
                        errordlg('calibration error');
                    end
                end
                % Slave1-----------------------------------------------------
                if (channelNum>=5 && channelNum <=8)
                    try
                        oldSyncMode     = get(userData.slave1,'SyncMode');

                        set(userData.slave1,...
                            'LogFileName'                   , ...
                                fullfile(userData.folderStr , ...
                                'DAQSession_Calibration'    , ...
                                ['Calibration_Channel_'     , ...
                                 num2str(channelNum,'%02d') , ...
                                 '_',userData.dateStr])     , ...
                            'SamplesAcquiredFcn'            , ...
                                {@getSensitivity_v2,...
                                 aInputObject,channelNum,oldSyncMode},...
                            'SyncMode'                      , 'None');
                        start(userData.slave1);
                    catch
                        errordlg('calibration error');
                    end
                end
                % Slave2-----------------------------------------------------
                if (channelNum>=9 && channelNum <=12)
                    try
                        oldSyncMode     = get(userData.slave2,'SyncMode');

                        set(userData.slave2,...
                            'LogFileName'                   , ...
                                fullfile(userData.folderStr , ...
                                'DAQSession_Calibration'    , ...
                                ['Calibration_Channel_'     , ...
                                 num2str(channelNum,'%02d') , ...
                                 '_',userData.dateStr])     , ...
                            'SamplesAcquiredFcn'            , ...
                                {@getSensitivity_v2,...
                                 aInputObject,channelNum,oldSyncMode},...
                            'SyncMode'                      , 'None');
                        start(userData.slave2);
                    catch
                        errordlg('calibration error');
                    end
                end
                % Slave3-----------------------------------------------------
                if (channelNum>=13 && channelNum <=16)
                    try
                        oldSyncMode     = get(userData.slave3,'SyncMode');

```

```
363                         set(userData.slave3,...
364                             'LogFileName'                   , ...
365                                 fullfile(userData.folderStr , ...
366                                 'DAQSession_Calibration'    , ...
367                                 ['Calibration_Channel_'     , ...
368                                  num2str(channelNum,'%02d') , ...
369                                  '_',userData.dateStr])     , ...
370                             'SamplesAcquiredFcn'            , ...
371                                 {@getSensitivity_v2,...
372                                  aInputObject,channelNum,oldSyncMode},...
373                             'SyncMode'                      , 'None');
374                         start(userData.slave3);
375                     catch
376                         errordlg('calibration error');
377                     end
378                 end
379         end
380     end
381 else
382     warndlg('Calibration unsuccesful!');
383 end
384 %-------------------------------------------------------------------------
385 %% Wegschreiben der Ergebnisse
386 set(aInputObject,'UserData',userData); % Objekt-Spezifische Daten
387 %-------------------------------------------------------------------------
388 end
```

```
1  function close_callback(gcobject)
2  %-------------------------------------------------------------------------
3  % Schnittstellenbeschreibung
4  %
5  % Autor:                Rick Plescher, 30.04.2015
6  %
7  % Funktionsaufruf:      callback-Event
8  %
9  % Fkt.-beschreibung:    Bei drücken des 'Close'-Button der MAMS-GUI wird
10 %                       diese Funktion aufgerufen.
11 %
12 % Eingabe:
13 %   gcobject:           gcobject ist ein handle vom aktuellen Objekt,
14 %                       welches diese callback-function aufgerufen hat.
15 %
16 % Ausgabe:              - keine Ausgabe -
17 %
18 %
19 % sontiges:
20 %
21 %-------------------------------------------------------------------------
22
23 %% Intialisierung
24
25 %-------------------------------------------------------------------------
26 %% Überprüfung der Eingabe-Parameter
27
28 %-------------------------------------------------------------------------
29 %% Programmcode
30
31 %-------------------------------------------------------------------------
32 %% Wegschreiben der Ergebnisse
```

```
33
34 %-----------------------------------------------------------------------
35 % userData = get(gcobject,'UserData');
36 delete(gcobject);
37 % delete(userData.hPlotRefSignal);
38
39 end
```

```
1  function out = closeCalibration_callback(aObject,event,aInput)
2  %-----------------------------------------------------------------------
3  % Schnittstellenbeschreibung
4  %
5  % Autor:                Rick Plescher, 30.04.2015
6  % Funktionsaufruf:  out = closeCalibration_callback(aObject,event,aInput)
7  %
8  % Fkt.-beschreibung: Stopt das ausführende Objekt
9  %
10 % Eingabe:
11 %
12 %
13 % Ausgabe:
14 %
15 %
16 % sontiges:
17 %
18 %-----------------------------------------------------------------------
19
20 %% Intialisierung
21
22 %-----------------------------------------------------------------------
23 %% Überprüfung der Eingabe-Parameter
24
25 %-----------------------------------------------------------------------
26 %% Programmcode
27 if strcmp(get(aInput,'Running'),'On')
28     stop(aInput);
29 end
30 %-----------------------------------------------------------------------
31 %% Wegschreiben der Ergebnisse
32
33 %-----------------------------------------------------------------------
34 end
```

```
1  function out = configureDT9837BChannels(aInput)
2  %-----------------------------------------------------------------------
3  % Schnittstellenbeschreibung
4  %
5  % Autor:                Rick Plescher, 30.04.2015
6  %
7  % Funktionsaufruf:      out = configureDT9837BChannels(aInput)
8  %
9  % Fkt.-beschreibung:    Diese Funktion initialisiert dens angegebenen Kanal
10 %                       am jeweiligen DAQ-Modul von Data Translation aus
11 %                       der DT9837B Produktgruppe.
12 %                       Üblicherweise wird die Funktion im Schleifen-
13 %                       Durchlauf aufgerufen. Daher sind zusätzliche
14 %                       Maßnahmen notwendig um doppelte EInträge zu
15 %                       vermeiden.
```

```
16 %
17 % Eingabe:
18 %   aInput               : Eingabestruktur
19 %       .n               : Kanalzahl des zu initialisierenden Kanal [1-4]
20 %       .module          : Struktur des zu konfiguriendenen DAQ-Modul
21 %       .microphoneData  : Daten aller angeschlossenen Mikrofone
22 %
23 % Ausgabe:
24 %   out                  : Ausgabestruktur
25 %       .module          : Struktur mit objekt-spezifischen Daten des DAQ-
26 %                          Modul mit neu initialisierten Kanal
27 %
28 % sontiges:
29 %   Für die richtige FUnktion muss sicher gestellt werden ,dass die Cell-
30 %   Struktur "microphoneData" ohne Header(keine Item-Bezeichnung in der
31 %   ersten Zeile) übergeben wird.
32 %
33 %-------------------------------------------------------------------------
34 %% Programmcode
35 % Prüfung, ob ein Mikrofon richtig konfiguriert wurde (hier wird geprüft ob
36 % ein Modell-Name vergeben wurde
37 if (strcmp(aInput.microphoneData{aInput.n,4},' ')==0)
38     % Ermittlung der angemeldeten Hardware-Kanäle (HW-Kanäle)
39     existingHwChannelIdx    = ...
40         get(get(aInput.module,'Channel'),'HwChannel');
41     % Wenn mehr als eine Kanal angemeldet ist muss die resultierenden
42     % Cell-Struktur noch in eine Matrix umgewandelt werden
43     if iscell(existingHwChannelIdx)
44         existingHwChannelIdx = ...
45             cell2mat(existingHwChannelIdx);
46     end
47
48     % Prüfung auf doppelte Einträge, Vergleich mit den HW-Kanälen in der
49     % globalen Mikrofon-Datenbank
50     doubleIdx = ...
51         find(existingHwChannelIdx==...
52         aInput.microphoneData{aInput.n,3});
53     % Wenn der HW-Kanal nicht bereits angemeldet wurde wird er jetzt
54     % hinzugefügt
55     if isempty(doubleIdx)
56         try
57             % der Kanal wird am DAQ-Modul angemeldet
58             tmpChan = ...
59                 addchannel(aInput.module,...
60                 aInput.microphoneData{aInput.n,3});
61
62             % Konfiguration des Kanals
63             % 'ExcitationCurrentSource' := Anschalten zur
64             %                              Konstant-Stromversorgung
65             % 'Coupling'                := AC => schaltetet einen Hochpass-
66             %                              Filter mit niedriger
67             %                              Grenzfrequenz an den Eingang um
68             %                              den Konstant-Strom zu filtern
69             % 'GainPerChan'             := Auf "1" gestellt ergibt sich ein
70             %                              Messbereich von [-10 V,10 V]
71             % 'ChannelName'             := String mit der Kanal-Bezeichnung
72
73             set(aInput.module.channel(get(tmpChan,'Index')),...
74                 'ExcitationCurrentSource'   ,'Internal',...
75                 'Coupling'                  ,'AC',...
76                 'GainPerChan'               , 1,...
```

```
                'ChannelName'                , ...
                ['Ch',num2str(aInput.n,'%02d')]);
        catch
            warndlg(['Can not initialize Channel ',...
                num2str(aInput.microphoneData{aInput.n,1})]);
        end
    end
end
%-------------------------------------------------------------------------
%% Wegschreiben der Ergebnisse
out = aInput.module;
%-------------------------------------------------------------------------
end
```

```
function displayGuiMicInfo()
%-------------------------------------------------------------------------
% Schnittstellenbeschreibung
%
% Autor:                Rick Plescher, 30.04.2015
%
% Funktionsaufruf:      displayGuiMicInfo()
%
% Fkt.-beschreibung:    Diese Funktion ruft beim Aufruf automatisch die
%                       aktuell Hinterlegten Mikrofondaten auf und stellt
%                       diese Übersichtlich dar.
%
% Eingabe:              keine Eingabe erforderlich
%
%
% Ausgabe:              keine direkte Ausgabe
%
%
% sontiges:
%   benötigt wird eine microphoneData.mat Datei im aktuellen Dateipfad
%-------------------------------------------------------------------------
%% Initialisierung
figureHandle = figure; % Erzeugung eines neuen Fenster
%-------------------------------------------------------------------------
%% Programmcode
set(figureHandle,...
    'Menubar'       , 'none',... % Ausschalten des Menüs
    'Name'          , 'Microphone Data',... % Anzeigen des Namens
    'NumberTitle'   , 'Off',... % Keine Nummerierung des Anzeigetitels
    'Resize'        , 'On'); % Veränderung der Größe mit der Maus Zulassen

screenUnits = 'pixels'; % Bildschirm-Messeinheit in Pixel
set(0,'Units',screenUnits); % Setzen der Bildschirm-Einehit
screenSize  = get(0,'ScreenSize'); % Einlesen der Bildschirmgröße

figWidth    = 0.70*screenSize(3); % Breite des Fensters definieren
figHeight   = 0.85*screenSize(4); % Höhe des Fensters definieren

set(figureHandle,... % Einstellen der neuen Fenstergröße und -position
    'Position',...
    [screenSize(3)/2-figWidth/2     , screenSize(4)/2-figHeight/2,...
    figWidth                        , figHeight]);

% Einlesen der Mikrofondaten+++++++++++++++++++++++++++++++++++++++++++++++
% aktuelle Mikrofondaten
```

```
46 readMicInput.file   = fullfile(cd,'microphoneData.mat');
47 readMicInput.header = true; % Ausgabe mit dem Header (Items)
48 % Einlesen der Daten als Cell-Array
49 data                = readMicrophoneData(readMicInput);
50 %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
51 columnname      = data(1,:); % Cell-Vektor mit den Items
52 columneditable  = false(size(columnname)); % Die Items können nicht
53                                            % editiert werden
54 % Definition des Format für die einzelnen Spalten der folgenden Tabelle
55 columnformat    = {...
56     'numeric'   ,...
57     'numeric'   ,...
58     'numeric'   ,...
59     'char'      ,...
60     'char'      ,...
61     'numeric'   ,...
62     'char'};
63 % Definition der Spaltenbreiten der folgenden Tabelle
64 columnWidth     = {50,80,100,180,120,120,270};
65 % Initialisierung der UI-Tabellen-Elements
66 uitable(...
67     'Units'          , 'normalized',... % Einheit der Tabellengröße/Position
68     'ColumnEditable', columneditable,... % Splaten sind editierbar
69     'ColumnFormat'   , columnformat,... % Format der Spalteneingabe
70     'ColumnName'     , columnname,... % Spaltenbezeichnung
71     'ColumnWidth'    , columnWidth,... % Spaltenbreite
72     'Data'           , data(2:end,:),... % Daten der Spalten
73     'FontSize'       , 20,... % Schriftgröße
74     'Parent'         , figureHandle,... % Einbettung im Handle des Fensters
75     'Position'       , [0 0 1 1]); % Position und Größe der Tabelle
76 %-------------------------------------------------------------------------
77 end
```

```
1 function editMicrophoneData()
2 %-------------------------------------------------------------------------
3 % Schnittstellenbeschreibung
4 %
5 % Autor:                 Rick Plescher, 30.04.2015
6 %
7 % Funktionsaufruf:       editMicrophoneData()
8 %
9 % Fkt.-beschreibung:     Die Funktion öffnet ein neues Fenster mit den
10 %                       bisher aktuell hinterlegten Mikrofondaten.
11 %                       Anschließend kann durch die Auswahl einer
12 %                       Seriennummer die Konfiguration der Mikrofone
13 %                       geändert werden. Das Mikrofon-Editier-Fenster
14 %                       ermöglicht auch das Laden oder Speichern von
15 %                       Mikrofon-Presets.
16 %
17 % Eingabe:               keine Eingabe erforderlich
18 %
19 % Ausgabe:               keine direkte Ausgabe
20 %
21 % sontiges:              Um die korrekte Funktionsweise zu ermöglichen muss
22 %                       eine aktuelel microphoneData.mat Datenbank im
23 %                       aktuellen Pfad hinterlegt sein und außerdem müssen
24 %                       folgende Funktionen im Matlab-Pfad zur Verfügng
25 %                       stehen:
26 %                       1. loadSavePresetMicData_keyCallback.m
```

```
%                        2. loadPresetMicData_mouseCallback.m
%                        3. savePresetMic_mouseCallback.m
%-------------------------------------------------------------------------
%% Intialisierung
f           = figure;  % Erzeugung eines neuen Fenster
%-------------------------------------------------------------------------
%% Programmcode
set(f,... % Konfiguration des neuen Fensters
    'Menubar'       ,'none',... % Menü ausschalten
    'Name'          ,'Choose your microphone setup',... % Name des Fensters
    'NumberTitle'   ,'Off',... % Keine Nummerierung des Fenster-Titels
    'Resize'        ,'Off'); % Keine Möglichkeit mit der Maus die
                             % Fenstergröße zu ändern

screenUnits = 'pixels'; % Einheit des Fenster := Pixel
set(0,'Units',screenUnits); % Einstellen der neuen Fenster-Einheit
screenSize  = get(0,'ScreenSize'); % Ermittlung der Bildschirmgröße

figWidth    = 0.60*screenSize(3); % Definition der Fensterbreite
figHeight   = 0.85*screenSize(4); % Definition der Fensterhöhe
set(f,... % Konfiguration der Position und der Größe des Fensters
    'Position',...
        [screenSize(3)/2-figWidth/2     , screenSize(4)/2-figHeight/2,...
         figWidth                       , figHeight]);
% Einlesen der Mikrofondata+++++++++++++++++++++++++++++++++++++++++++++++
% An dieser Stelle werden die Mikrofondaten ohne Header eingelesen. Die
% eingelesenen Daten stammen aus der aktuellsten Mikrofondatenbank im
% aktuellen Dateipfad
readMicInput.file   = fullfile(cd,'microphoneData.mat'); % Dateipfad
readMicInput.header = false; % Keine Items einlesen
data                = readMicrophoneData(readMicInput);% Einlesen der Daten
data                = data(:,4:end); % Es werden für jedes Mikrofon nur die
                                     % letzten 4 Spalten eingelesen
                                     % (Modell, Seriennummer,
                                     % Übertragungsfaktor, Datum der
                                     % letzten Kalibrierung
% An dieser Stelle wird die gesamte Datenbank der Mikrofone die jemals für
% das Messsystem eingetragen wurde abgelegt. Es können nur Mikrofone
% verwendet werden die in dieser Datenbank enthalten sind. Es ähnelt in der
% Verwendung einem Preset.

readMicInput.file   = ... % Dateipfad
    fullfile(cd,'MicrophoneDatabase',...
    'completeMicDatabase-DO NOT EDIT.mat');
readMicInput.header = true; % Einlesen inklusive des Headers
tmp                 = readMicrophoneData(readMicInput);% Einlesen der Daten
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% Darstellung der Mikrofondaten und Definition der Callback-Funktion+++++++
rowName             = {1:size(data,1)};
columnname          = ... % Spaltenbezeichnung
    {'Model', 'SN'                  , 'Sensitivity' , 'Last Calibration'};
columneditable      = ... % Einstellen der Editierbarkeit der Spalten
    [false  , true                  , false         , false];
columnformat        = ... % Format der Spalten
    {'char' , [tmp(2:end,5);{' '}]' , 'numeric'     , 'char'};
columnWidth         = ... % Spaltenbreite
    {180    , 180                   , 180           , 250};

% Initialisierung und Konfiguration der Tabelle mit den Mikrofondaten++++++
% Die CellEditCallback - Funktion wird immer ausgeführt wenn in der Tabelle
% neue Mikrofondaten vom Benutzer eingegeben werden. Damit wird die
```

```
88  % derzeitige microphoneData.mat aktualisiert.
89  t = uitable(... % Eigenschaften siehe: help Uitable Properties
90      'Units'              , 'normalized',...
91      'CellEditCallback'   , {@refreshMicData_callback},...
92      'ColumnEditable'     , columneditable,...
93      'ColumnFormat'       , columnformat,...
94      'ColumnName'         , columnname,...
95      'ColumnWidth'        , columnWidth,...
96      'Data'               , data,...
97      'FontSize'           , 20,...
98      'Position'           , [0 0 1 1],...
99      'RowName'            , rowName);
100 % Einlesen der geänderten Mikrofondaten in das Fenster-Handle und
101 % Definition des Key-Callback für das Laden und Speichern des Presets der
102 % Mikrofon-Daten
103 set(f,...
104     'UserData'       ,get(t,'UserData'),...
105     'KeyPressFcn'    ,{@loadSavePresetMicData_keyCallback,t});
106 %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
107 % Definiton des Save/Load Buttons und der Callback-Funktionen+++++++++++++
108 % Beim Maus-Klick auf den Load-Button wird ein Callback ausgeführt
109 hLoad = uicontrol(...
110     'Units'            , 'normalized',...
111     'Callback'         , {@loadPresetMicData_mouseCallback,t},...
112     'Parent'           , f,...
113     'Position'         , [0.1150 0.01 0.2 0.0556],...
114     'String'           , 'Load Preset',...
115     'Style'            , 'pushbutton',...
116     'Value'            , 1);
117 % Beim Maus-Klick auf den Save-Button wird ein Callback ausgeführt
118 hSave = uicontrol(...
119     'Units'            , 'normalized',...
120     'Callback'         , {@savePresetMicData_mouseCallback,t},...
121     'Parent'           , f,...
122     'Position'         , [0.4150 0.01 0.2 0.0556],...
123     'Style'            , 'pushbutton',...
124     'String'           , 'Save Preset',...
125     'Value'            , 1);
126 %-------------------------------------------------------------------------
127 %% Wegschreiben der Ergebnisse
128 waitfor(f); % In der Funktion wird so lange gewartet bis das Fenster
129             % geschlossen
130 %-------------------------------------------------------------------------
131 end
```

```
1  function getSensitivity_v2(aObject,~,aFigureHandle,aChannelNum,aOldSyncMode)
2  %-------------------------------------------------------------------------
3  % Schnittstellenbeschreibung
4  %
5  % Autor:                Rick Plescher, 30.04.2015
6  %
7  % Funktionsaufruf:      getSensitivity_v2(aObject,~,aFigureHandle,...
8  %                           aChannelNum,aOldSyncMode)
9  %
10 % Fkt.-beschreibung:
11 %
12 % Eingabe:
13 % aObject               : Handle des jeweiligen aufrufenden DAQ-Moduls
14 % aFigureHandle         : Handle der MAMS-GUI
```

```
15 % aChannelNum           : Nummer des zu kalibrierenden Kanals
16 % aOldSyncMode          : ursprünglicher Sync-Modus des DAQ-Moduls
17 %                         Für DT9837B z.B 'None', 'Master', 'Slave'
18 %
19 % Ausgabe:
20 %
21 %
22 % sontiges:
23 %
24 %-------------------------------------------------------------------------
25 %% Intialisierung
26 persistent counter; % Aufruf-Zähler (wie oft wurde die Funktion aufgerufen)
27 persistent lookBackCounter; % Zähler der gültigen Mikrofon-Empfindlichkeit
28 persistent sensitivity; % Mikrofon-Empfindlichkeit (Übertragungsfaktor)
29 persistent levelHistoryDB; % Historie der kalibrierten Pegel [dB] mit altem
30 persistent levelHistoryPa; % Historie der kalibrierten Pegel [Pa] ohne
31                           % Übertragungsfaktor
32 if isempty(counter) % bei der allerersten Aufruf werden die Zähler
33                     % initialisiert
34
35     lookBackCounter = 0;
36     counter         = 1;
37     sensitivity     = [];
38     levelHistoryDB  = [];
39     levelHistoryPa  = [];
40 else
41
42     counter = counter +1;
43
44 end
45
46
47 calibrationLevel    = 94;   % Kalibrierpegel
48 refPressure         = 2E-5; % Bezugsgröße für den Schalldruck [dB]
49 calibrationPressure = refPressure*10^(calibrationLevel/20); % Bezugsgröße
50                                                   % für den Schalldruck [Pa]
51 maxCalibVariance    = 0.5; % maximal zulässige Abweichung vom
52                            % Kalibrierpegel in dB
53 %-----------------------------------------------------------------------
54 %% Programmcode
55 userData        = get(aFigureHandle,'UserData'); % Handle der MAMS-GUI
56 numSamples      = get(aObject,'SamplesAcquiredFcnCount'); % Anzahl der
57  % Samples ab wann die Kalibrierungsfunktion aufgerufen wird
58  % entspricht auch der Blocklänge der normalen Datenerfassung
59 sampleRate      = get(aObject,'SampleRate'); % Einlesen der Abtastrate
60 sensBackLookIdx = ceil(5*sampleRate/numSamples); % Anzahl der Samples über
61 % die der Kalibrier-Ton innerhalb der zulässigen Abweichung vom
62 % Referenz-Pegel liegen soll (Hier ==> 5 Sekunden)
63 % Einlesen der Mikrofondaten-----------------------------------------------
64 readMicInput.file       = fullfile(fullfile(cd,'microphoneData.mat'));
65 readMicInput.header     = false;
66 userData.microphoneData = readMicrophoneData(readMicInput);
67 oldMicSensitivity       = userData.microphoneData{aChannelNum,6};
68 if oldMicSensitivity == 0 % Falls noch keine Sensitivity vorhanden ist
69     % wird standardmäßig mit 50 mV/Pa gerechnet
70     oldMicSensitivity = 50e-3;
71 end
72 % Prüfen ob das Kalibrier-Fenster noch da ist-----------------------------
73 if ishandle(userData.calibrationWindowStruct.plotHandle(2))
74     data            = peekdata(aObject,numSamples); % einlesen der
75     % aktuellen Messdaten (je nach Anzahl der aktiven Kanäle am Modul kan
```

```
76      % es bis zu 4 Spalten haben)
77      hwChannelNum     = userData.microphoneData{aChannelNum,3}; % Einlesen
78      % des aktuellen Hardware-Kanal des DAQ-Modul
79
80      % Ermittlung des Kanal-Idx zum Auslesen des richtigen Kanals in der
81      % Daten-Matrix
82      if iscell(aObject.Channel.HwChannel)
83
84          channelIdx  = ...
85              aObject.Channel.Index(...
86              cell2mat(aObject.Channel.HwChannel)==hwChannelNum);
87
88      else
89
90          channelIdx = aObject.Channel.Index(1);
91
92      end
93
94      % Hinweis-------------------------------------------------------------
95      % Unterschiede zu den verschiedenen Kanal-Arten
96      %   aChannelNum := Kanal-Nummer von 1 - 16 (maximal 16 Kanäle)
97      %   hwChannelNum:= modul spezifische Kanal-Nummer von 0 - 3
98      %   (repräsentiert die Kanäle 1-4)
99      %   channelIdx  := Kanal-Zahl zum Einlesen der Daten (1-4)
100     %-------------------------------------------------------------------
101     % Kalibrierungs-Algorithmus-------------------------------------------
102     % Fensterung inklusive Amplituden-Korrektur+++++++++++++++++++++++++
103     % 1. Berechnung des Gewichtungs-Fenster für die Fensterung des
104     % erfassten Messdaten-Block
105     winFFT           = flattopwin(length(data(:,channelIdx)));
106     % 2. Fensterung des aktuellen Messdaten-Block
107     sigWin           = data.*winFFT*(length(data)/sum(winFFT));
108     % length(data)/sum(winFFT) ist Amplitudenkorrektur für die
109     % Energieänderung durch das Fenster
110     %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
111     % 3. Berechnung der komplexen Fourier-Transformieten+++++++++++++++++
112     dataComplexFFT   = fft(sigWin);
113     %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
114     % 4. Berechnung des Amplitudenspektrum++++++++++++++++++++++++++++++
115     dataAbsFFT       = abs(dataComplexFFT);
116     %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
117     % 5.Normierung des Spektrum und Beschränkung auf das Einseiten-Spektrum
118     dataAbsFFT       = 2*dataAbsFFT(1:end/2)/length(data); % Berechnung des
119     % 6. Einseiten-Spektrums
120     dataAbsFFT(1)    = dataAbsFFT(1)/2; % Halbierung des ersten Bins, weil
121     % die Spektrallinien bei 1 und NFFT/2 zusammen fallen
122     % Berechnung der Effektiv-Werte+++++++++++++++++++++++++++++++++++++
123     dataRMSFFT       = dataAbsFFT/sqrt(2); % Berechnung des Effetivwertes
124     % ==> Crest-Faktor für Sinus-Schwingungen 1/sqrt(2) ~ 0,707
125     %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
126     df               = sampleRate/length(data); % Berechnung der
127     % Frequenzauflösung der FFT
128     freqRange        = 0:df:sampleRate/2-1; % Berechnung des Frequenz-Vektor
129
130     % Gültiger Frequenzbereich in den der Kalibrier-Ton liegen darf
131     calibratorFreqRangeIdx  = find(freqRange>=960 & freqRange<=1040);
132
133     % Suche nach dem Pegel des Kalibrier-Ton im zulässigen Bereich
134     [maxFFTValue,maxFFTIdx] = max(dataRMSFFT(calibratorFreqRangeIdx));
135     % Umrechnung des Index in eine Frequenz
136     maxFFTFreqBin            = freqRange(calibratorFreqRangeIdx(maxFFTIdx));
```

```
137     % Umrechnung des Index des Kalibrierbereich auf den gesamten
138     % Frequenzbereich
139     maxFFTIdx                = calibratorFreqRangeIdx(maxFFTIdx);
140     % Berechnung der aktuellen Mikrofon-Empfindlichkeit++++++++++++++++++
141     sensitivity(counter)     = dataRMSFFT(maxFFTIdx)/calibrationPressure;
142     % Berechnung der FFT kalibriert auf den alten Kalibierton
143     dataDBFFT                = ...
144         20*log10((dataRMSFFT/oldMicSensitivity)/refPressure);
145     % Speichern des aktuellen Kalibierpegels in die Historie
146     levelHistoryDB(counter) = dataDBFFT(maxFFTIdx);
147     levelHistoryPa(counter) = dataRMSFFT(maxFFTIdx);
148     %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
149     % Kurve für die FFT wird rot-gezeichnet, und die Daten gezeichnet
150     set(userData.calibrationWindowStruct.plotHandle(2),...
151         'Color'      , 'red',...
152         'XData'      ,freqRange,...
153         'YData'      ,dataDBFFT);
154     % Anzeige des neuen Titels
155     set(userData.calibrationWindowStruct.titleHandle(2),...
156         'String'     ,['DFT des Eingangssignals Block-Nr.: ',...
157         num2str(counter)]);
158     % Anzeige der aktuellen Informationen des Blocks
159     set(userData.calibrationWindowStruct.txtDisplayHandle(1),...
160         'String'     , {'max Value: ';...
161         [num2str(20*log10(maxFFTValue/oldMicSensitivity/refPressure),...
162         '%03.1f'),' dB @ '];...
163         [num2str(maxFFTFreqBin,'%04.2f'),' Hz'];...
164         'calc.Sens.: ';...
165          [num2str(sensitivity(counter)*1e3,'%02.2f'),'mV/Pa']});
166
167     drawnow; % Aktualisierung des GUI-Fensters
168
169     % Sobald der Zähler Counter größer als der SensBackLookIdx ist die
170     % Mindestwartezeit für ein stabiles Signal vorbeis.
171     if counter>=sensBackLookIdx
172         % Berechnung der relelativen Abweichung der letzten Sekunden des
173         % Kalibriertons
174         tmp                  = max(abs(diff(...
175             levelHistoryDB(end-sensBackLookIdx+1:end))));
176        % Abfrage: wenn die relativen Abweichungen der letzten Sekunden
177        % maximal der zulässigen Abweichung entspricht wird die zweite
178        % Kalibirschleife begonnen
179        if tmp <= maxCalibVariance
180             % Beginn der zweiten Kalibrierschleife
181             % Kurve des Kalibrier-Fenster wird grün dargestellt
182             set(userData.calibrationWindowStruct.plotHandle(2),...
183             'Color'      , 'green');
184
185         % Wenn der Index größer als die Mindest-Wartezeit ist ist die
186         % Kalibrierung erfolgriech := Diese Schleife sorgt dafür, dass ide
187         % Kalibrierung erst dann erfolgt, wenn eine dauerhafter konstanter
188         % Pegel gemessen wird
189             if lookBackCounter >= sensBackLookIdx
190
191                 stop(aObject); % Stop der Datenerfassung
192                 % set(aObject,'SyncMode',aOldSyncMode);
193
194                 newMicSensitivity   = sensitivity(end); % Auswahl des
195                 % Mikrofon-Übertragungsfaktor
196
197                 % Berechnung des Unterschieds zwischen den Pegel mit neuem
```

```
198                 % und alten Differenzpegel
199                 levelDifference     = ...
200                     20*log10(levelHistoryPa(end)/...
201                     newMicSensitivity/refPressure)-...
202                     20*log10(levelHistoryPa(end)/...
203                     oldMicSensitivity/refPressure);
204                 % Benutzer-Abfrage ob der neue Mikrofon-Übertragunsfaktor
205                 % übernommen werden soll
206                 acceptCalibrationButton  =...
207                     questdlg(...
208                     {'Do you to save the new sensitivity?';...
209                      ['old Sensitivity: ',...
210                       num2str(oldMicSensitivity*1E3,'%02.2f')];...
211                      ['new Sensitivity: ',...
212                       num2str(newMicSensitivity*1E3,'%02.2f')];...
213                      ['Level Difference to old sensitivity: ',...
214                       num2str(levelDifference,'%03.1f')]},...
215                     'New Sensitivity',...
216                     'Yes',...
217                     'No',...
218                     'No');
219
220                 % Wenn der Übertragungsfaktor übernommen wird wird er im
221                 % aktuellen Mirkofon-datenfile übernommen
222
223                 if strcmp(acceptCalibrationButton,'Yes')
224
225                 userData.microphoneData{aChannelNum,6} = newMicSensitivity;
226                 userData.microphoneData{aChannelNum,7} = datestr(now,30);
227
228                 microphoneItems = { ...
229                     'Num'           ,'Ch-Num'      ,'HwCh-Num' ,'Model',...
230                     'Serialnumber'  ,'Sensitivity' ,'Last Calibration'};
231                 userData.microphoneData  =...
232                     [microphoneItems;userData.microphoneData];
233                 microphoneData           = ...
234                     userData.microphoneData;
235                 % Speichern der Mikrofon-Daten
236                 save(fullfile(cd,filesep,'microphoneData.mat'),...
237                     'microphoneData');
238                 else
239
240                 end
241
242                 helpdlg('Calibration Finished!');
243                 lookBackCounter = 0;
244                 counter         = 1;
245                 sensitivity     = [];
246                 levelHistoryDB  = [];
247                 levelHistoryPa  = [];
248
249             end
250
251             lookBackCounter = lookBackCounter+1;
252         else % Falls die relativ Abweichung vom Kalibrier-Ton zu groß war
253             % wird der entsprechende Counter zurückgesetzt
254
255             set(userData.calibrationWindowStruct.plotHandle(2),...
256                 'Color'     , 'red');
257             lookBackCounter = 0;
258         end
```

```
    end
else
    lookBackCounter = 0;
    counter         = 1;
    sensitivity     = [];
    levelHistoryDB  = [];
    levelHistoryPa  = [];

    stop(aObject);
    set(aObject,'SyncMode',aOldSyncMode);
end
%-------------------------------------------------------------------------
%% Wegschreiben der Ergebnisse
% set(userData.calibrationWindowStruct.hCalibrationHandle,...
%     'DeleteFcn'         ,{@closeCalibration_callback,aObject});
set(aFigureHandle,'UserData',userData);
%-------------------------------------------------------------------------
end
```

```
function  out = initDaqSetup(aInput)
%-------------------------------------------------------------------------
% Schnittstellenbeschreibung
%
% Autor:                Rick Plescher, 30.04.2015
%
% Funktionsaufruf:      out = initDaqSetup(aInput);
%
% Fkt.-beschreibung:    Diese Funktion konfiguriert die angeschlossenen
%                       DAQ-Module und die aktiven Kanäle.
%
% Eingabe:
%  aInput
%   .hInfoText          : Handle zu de Info-Textbox
%   .hMamsGui           : Handle zu der grafischen Benutzeroberfläche des
%                         Datenerfassungssystems
%
% Ausgabe:
%   out
%    .blockLengthDAQ    : Blocklänge [Samples]
%    .folderStr         : Speicherort [String]
%    .masterIdx         : Index des angeschlossenen Slave Idx
%    .microphoneData    : Mikrofondaten [cell-struct]
%    .moduleString      : Specher-Name [String]
%    .numActiveChannels : Anzahl der aktiven Kanäle
%    .numDaqModules     : Anzahl der angeschlossenen
%    .sampleRate        : Abtastrate der DAQ-Module [double]
%    .slaveIdx          : DAQ-Slave Index-Array [double array]
%
% sontiges:
%   Voraussetzung für eine erfolgreiche Konfiguration ist die Verwendung
%   einer 32-Bit Version von Matlab und der Data Translation Open Layer
%   (dtol)-Adapter muss instaliert sein
%
%   notwendige Funktionen:
%       1. configureDT9837BChannels.m
%       2. editMicrophoneData.m
%       3. readMicrophoneData.m
%-------------------------------------------------------------------------
%% Intialisierung
```

```
41 out              = struct();
42 connectedDevices= struct();
43 listString       = cell(8,1);
44 moduleString     = {...
45     'DAQSession_Master';...
46     'DAQSession_Slave1';...
47     'DAQSession_Slave2';...
48     'DAQSession_Slave3'};
49 %-------------------------------------------------------------------------
50 %% Benutzer-Abfrage zur Abtastraten-Konfiguration
51 listString(1)    = ...
52     {['Samplerate = 4 kHz/Samples per Block (online-View) = ',...
53     ' 1024/frequency Resolution (online-FFT)  = 3,906 Hz/Bin']};
54 listString(2)    = ...
55     {['Samplerate = 8 kHz/Samples per Block (online-View) = ',...
56     '2048/frequency Resolution (online-FFT)   = 3,906 Hz/Bin']};
57 listString(3)    = ...
58     {['Samplerate = 10 kHz/Samples per Block (online-View) = ',...
59     '2048/frequency Resolution (online-FFT)   = 4,883 Hz/Bin']};
60 listString(4)    = ...
61     {['Samplerate = 22.05 kHz/Samples per Block (online-View) = ',...
62     '4096/frequency Resolution (online-FFT) = 5,383 Hz/Bin']};
63 listString(5)    = ...
64     {['Samplerate = 32 kHz/Samples per Block (online-View) = ',...
65     '8192/frequency Resolution (online-FFT)   = 3,906 Hz/Bin']};
66 listString(6)    = ...
67     {['Samplerate = 44.1 kHz/Samples per Block (online-View) = ',...
68     '8192/frequency Resolution (online-FFT)   = 5,383 Hz/Bin']};
69 listString(7)    = ...
70     {['Samplerate = 48 kHz/Samples per Block (online-View) = ',...
71     '8192/frequency Resolution (online-FFT)   = 5,859 Hz/Bin']};
72 listString(8)    = ...
73     {['Samplerate = 96 kHz/Samples per Block (online-View) = ',...
74     '16384/frequency Resolution (online-FFT)  = 5,859 Hz/Bin']};
75
76 sampleRateConfig= ... % Aufruf der Listen-Dialogs
77     listdlg(...
78     'fus'            , 1         ,...
79     'ffs'            , 1         ,...
80     'InitialValue'   , 8         ,...
81     'ListSize'       ,[600,150]  ,...
82     'ListString'     ,listString ,...
83     'PromptString'   ,'Which Samplerate do you want to use?',...
84     'SelectionMode' ,'Single'    ,...
85     'uh'             ,           30);
86
87 if isempty(sampleRateConfig) % Default-Wert: fs = 48 kHz;8192 Samples
88     sampleRateConfig = 8;
89 end
90
91 switch(sampleRateConfig)
92     case 1
93         sampleRate      = 4E3;
94         blockLengthDAQ  = 1024;
95     case 2
96         sampleRate      = 8E3;
97         blockLengthDAQ  = 2048;
98     case 3
99         sampleRate      = 10E3;
100        blockLengthDAQ  = 2048;
101    case 4
```

```
102         sampleRate      = 22.05E3;
103         blockLengthDAQ  = 4096;
104     case 5
105         sampleRate      = 32E3;
106         blockLengthDAQ  = 8192;
107     case 6
108         sampleRate      = 44.1E3;
109         blockLengthDAQ  = 8192;
110     case 7
111         sampleRate      = 48E3;
112         blockLengthDAQ  = 8192;
113     case 8
114         sampleRate      = 96E3;
115         blockLengthDAQ  = 16384;
116
117     otherwise
118         sampleRate      = 48E3;
119         blockLengthDAQ  = 8192;
120 end
121 %-------------------------------------------------------------------------
122 %% Benutzer-Abfrage zu den DAQ-Modulen
123 %Berechnung des notwendigen Bufferspeichers
124 lengthBuffer    = ceil(2*sampleRate/blockLengthDAQ);
125 if lengthBuffer < 2
126     lengthBuffer = 2; %internes Minimum bei manueller Konfiguration
127 end
128 % % folderStr       = fullfile('F:',datestr(now,29)); % Auf Privat-Rechner
129 folderStr       = fullfile('D:',datestr(now,29)); % Auf dem MAMS-Rechner
130 hardwareInfo    = daqhwinfo('dtol');
131 displayGuiTextInfo(aInput.hInfoText,02);
132 % % hardwareInfo    = daqhwinfo('winsound'); % angeschlossene Geräte suchen
133 % % displayGuiTextInfo(aInput.hInfoText,02); % Statusmeldung ausgeben
134 % Ermittlung der Anzahl der angeschlossenenen Module
135 numDaqModules   = size(hardwareInfo.BoardNames,2);
136 displayGuiTextInfo(aInput.hInfoText,03); % Statusmeldung ausgeben
137
138 masterIdx       = listdlg(... % Aufruf des Listen-Dialogs
139     'fus'           , 1         ,...
140     'ffs'           , 1         ,...
141     'ListSize'      , [320,260] ,...
142     'PromptString'  , 'Please Select the Master-Module!',...
143     'SelectionMode' , 'Single'  ,...
144     'ListString'    , hardwareInfo.BoardNames);
145
146 slaveIdx        = []; % Initialisierung des Slave-Vektros
147
148 % Ermittlung der Slave-Hardware-Indizes
149 if numDaqModules > 1
150     tmp         = 1:numDaqModules; % Vektor von 1 bis Anzahl DAQ-Module
151     tmpIdx      = tmp(tmp~=masterIdx); % Vektor ohne Master-Index
152     slaveIdx    = zeros(1,length(tmp)); % Slave-Index Vektor initialisieren
153
154     for n=1:length(tmpIdx)
155         tmpSlaveIdx             =...
156             str2double(hardwareInfo.BoardNames{tmpIdx(n)}(end-3));
157         slaveIdx(tmpSlaveIdx)   = tmpIdx(n);
158     end
159 end
160
161 displayGuiTextInfo(aInput.hInfoText,04);% Statusmeldung ausgeben
162
```

```
163 % Anlegen eines Ordner zum Abspeichern der Mikrofondaten
164 if exist(fullfile(folderStr,'DAQSession_MicrophoneData'),'dir')~=7
165     mkdir(fullfile(folderStr,'DAQSession_MicrophoneData'));
166 end
167
168 %Anlegen eines Ordner zum Abspeichern der Kalibrierungsdatei
169 if exist(fullfile(folderStr,'DAQSession_Calibration'),'dir')~=7
170     mkdir(fullfile(folderStr,'DAQSession_Calibration'));
171 end
172
173 % Anlegen eines Ordners zum Abpseichern der Event-Daten
174 if exist(fullfile(folderStr,'DAQSession_LogData'),'dir')~=7
175     mkdir(fullfile(folderStr,'DAQSession_LogData'));
176 end
177
178 % Anlagen eines Ordners zum Abspeichern der Messdaten
179 for n=1:numDaqModules
180     if exist(fullfile(folderStr,moduleString{n}),'dir')~=7
181         mkdir(fullfile(folderStr,moduleString{n}));
182     end
183 end
184
185 displayGuiTextInfo(aInput.hInfoText,05);% Statusmeldung ausgeben
186 %% Einlesen der zuvor angegebenen Mikrofondaten
187 readMicInput.file   = fullfile(cd,'microphoneData.mat');
188 readMicInput.header = false;
189 % Einlesen der zugehörigren Mikrofondatei
190 microphoneData      = readMicrophoneData(readMicInput);
191 numActiveChannels   = size(microphoneData,1);
192 %-------------------------------------------------------------------------
193 %% Programmcode
194 % Switch-Abfrage für die verschiedene Anzahl von angeschlossenen
195 % DAQ-Modulen. Es gibt die Möglichleit ziwschen 1 und 4 Module
196 % anzuschließen. Jeder dieser Fälle wird mit dieser Fallunterscheidung
197 % abgefragt.
198 switch numDaqModules
199     case 1 % Ein DAQ-Modul angeschlossen: Master
200         try
201             displayGuiTextInfo(aInput.hInfoText,06);
202             master = analoginput('dtol',...
203                 hardwareInfo.InstalledBoardIds{masterIdx});
204             % %             master = ...
205             % %                 analoginput('winsound',...
206             % %                 hardwareInfo.InstalledBoardIds{masterIdx});
207         catch
208             displayGuiTextInfo(aInput.hInfoText,81);
209         end
210
211         userDateStr = datestr(now,30); % Datum-String nach ISO 8601
212
213         displayGuiTextInfo(aInput.hInfoText,07);
214
215         % Konfiguration des Master-Module----------------------------------
216         % BufferingConfig           := [Blocklänge,Anzahl der Blöcke]
217         % SamplesAcquiredFcnCount   := Anzahl der Samples ab wann die
218         %                              SamplesAcquiredFcn ausgeführt wird
219         % SamplesPerTrigger         := Anzahl der zu erfassenden Samples
220         %                              bei ausgelöstem Trigger
221         % SampleRate                := Abtastrate
222         % LogToDiskMode             := Speicherungs-Modus (Datei-Name)
223         % LoggingMode               := Speicherungs-Modus (Dateiort)
```

```
224         % TimeOut                    := maximale Ausführzeit
225         % TriggerRepeat              := Anzahl der zu wiederholenden Trigger
226         % TriggerType                := Trigger-Typ
227         set(master,...
228             'BufferingConfig'        , [blockLengthDAQ, lengthBuffer],...
229             'SamplesAcquiredFcnCount', blockLengthDAQ,...
230             'SamplesPerTrigger'      , Inf           ,...
231             'SampleRate'             , sampleRate    ,...
232             'LogToDiskMode'          , 'Index'       ,...
233             'LoggingMode'            , 'Disk'        ,...
234             'TimeOut'                , 2             ,...
235             'TriggerRepeat'          , 0             ,...
236             'TriggerType'            , 'Immediate'   );
237
238         set(master,...
239             'SyncMode'                  , 'Master',...
240             'SamplesAcquiredFcn'        , {@plotData1,aInput.hMamsGui},...
241             'LogFileName'               , ...
242             fullfile(folderStr,'DAQSession_Master',userDateStr));
243 % %         set(master,...
244 % %             'SamplesAcquiredFcn'        , {@plotData1,aInput.hMamsGui},...
245 % %             'LogFileName'               , ...
246 % %                 fullfile(folderStr,'DAQSession_Master',userDateStr));
247         % Konfiguration der Kanäle des Master-Module-----------------------
248 % %         connectedDevices.master =...
249 % %                     configureDUMMYChannel(master);
250         for n= 1:numActiveChannels
251             if microphoneData{n,1} <= 4
252                 configureInput.module           = master;
253                 configureInput.microphoneData   = microphoneData;
254                 configureInput.n                = n;
255
256                 connectedDevices.master =...
257                     configureDT9837BChannels(configureInput);
258             end
259         end
260     case 2 % Zwei DAQ-Module angeschlossen: Master-Slave
261         try %Versuch der Anmeldung von zwei Modulen
262             displayGuiTextInfo(aInput.hInfoText,06);
263             master = analoginput('dtol',...
264                 hardwareInfo.InstalledBoardIds{masterIdx});
265             slave1 = analoginput('dtol',...
266                 hardwareInfo.InstalledBoardIds{slaveIdx});
267         catch % Falls das nicht funktioniert gibt es eine Fehlermeldung
268             displayGuiTextInfo(aInput.hInfoText,81);
269         end
270
271         userDateStr = datestr(now,30);
272
273         displayGuiTextInfo(aInput.hInfoText,07);
274
275         % Konfiguration des Mater- und Slave-Modul
276         set([master,slave1],...
277             'BufferingConfig'        , [blockLengthDAQ, lengthBuffer],...
278             'SamplesAcquiredFcnCount', blockLengthDAQ,...
279             'SamplesPerTrigger'      , Inf           ,...
280             'SampleRate'             , sampleRate    ,...
281             'LogToDiskMode'          , 'Index'       ,...
282             'LoggingMode'            , 'Disk'        ,...
283             'TimeOut'                , 2             ,...
284             'TriggerRepeat'          , 0             ,...
```

```
            'TriggerType'             , 'Immediate'   );
        % Kofiguration des Sync-Modus und der Callback-Funktion------------
        set(master,...
            'SyncMode'                  , 'Master' ,...
            'SamplesAcquiredFcn'        , {@plotData1,aInput.hMamsGui},...
            'LogFileName'               , ...
            fullfile(folderStr,'DAQSession_Master',userDateStr));

        set(slave1,...
            'SyncMode'                  , 'Slave'       ,...
            'SamplesAcquiredFcn'        , {@plotData2,aInput.hMamsGui},...
            'LogFileName'               , ...
            fullfile(folderStr,'DAQSession_Slave1',userDateStr));
        % Konfiguration der aktiven Kanäle
        for n = 1:numActiveChannels % Schleife über alle aktiven Kanäle
            % Konfiguration der aktiven Kanäle (Master-Modul CH = 1:4)
            if microphoneData{n,1} <= 4
                configureInput1.module           = master;
                configureInput1.microphoneData   = microphoneData;
                configureInput1.n                = n;


                connectedDevices.master =...
                    configureDT9837BChannels(configureInput1);
            end
            % Konfiguration der aktiven Kanäle (Slave1-Modul CH = 5:8)
            if ((microphoneData{n,1} >4) && (microphoneData{n,1}<=8))
                configureInput2.module           = slave1;
                configureInput2.microphoneData   = microphoneData;
                configureInput2.n                = n;

                connectedDevices.slave1...
                    = configureDT9837BChannels(configureInput2);
            end
        end

    case 3 % Drei DAQ-Module angeschlossen: Master-Slave-Slave
        try % Versuch der Anmeldung von drei Modulen
            displayGuiTextInfo(aInput.hInfoText,06);
            master = analoginput('dtol',...
                hardwareInfo.InstalledBoardIds{masterIdx});
            slave1 = analoginput('dtol',...
                hardwareInfo.InstalledBoardIds{slaveIdx(1)});
            slave2 = analoginput('dtol',...
                hardwareInfo.InstalledBoardIds{slaveIdx(2)});
        catch % Falls das nicht funktioniert gibt es eine Fehlermeldung
            displayGuiTextInfo(aInput.hInfoText,81);
        end

        userDateStr = datestr(now,30);

        displayGuiTextInfo(aInput.hInfoText,07);
        % Konfiguration der DAQ-Moduke-------------------------------------
        set([master,slave1,slave2],...
            'BufferingConfig'        , [blockLengthDAQ, lengthBuffer],...
            'SamplesAcquiredFcnCount', blockLengthDAQ,...
            'SamplesPerTrigger'      , Inf           ,...
            'SampleRate'             , sampleRate    ,...
            'LogToDiskMode'          , 'Index'       ,...
            'LoggingMode'            , 'Disk'        ,...
            'TimeOut'                , 2             ,...
```

```
            'TriggerRepeat'          , 0              ,...
            'TriggerType'            , 'Immediate'    );
        % Kofiguration des Sync-Modus und der Callback-Funktion------------
        set(master,...
            'SyncMode'                  , 'Master' ,...
            'SamplesAcquiredFcn'        , {@plotData1,aInput.hMamsGui},...
            'LogFileName'               , ...
            fullfile(folderStr,'DAQSession_Master',userDateStr));

        set(slave1,...
            'SyncMode'                  , 'Slave'       ,...
            'SamplesAcquiredFcn'        , {@plotData2,aInput.hMamsGui},...
            'LogFileName'               , ...
            fullfile(folderStr,'DAQSession_Slave1',userDateStr));

        set(slave2,...
            'SyncMode'                  , 'Slave'       ,...
            'SamplesAcquiredFcn'        , {@plotData3,aInput.hMamsGui},...
            'LogFileName'               , ...
            fullfile(folderStr,'DAQSession_Slave2',userDateStr));

        % Konfiguration der aktiven Kanäle
        for n= 1:numActiveChannels % Schleife über alle aktiven Kanäle
            % Konfiguration der aktiven Kanäle (Master-Modul CH = 1:4)
            if microphoneData{n,1} <= 4
                configureInput1.module           = master;
                configureInput1.microphoneData   = microphoneData;
                configureInput1.n                = n;

                connectedDevices.master =...
                    configureDT9837BChannels(configureInput1);
            end

            % Konfiguration der aktiven Kanäle (Slave1-Modul CH = 5:8)
            if ((microphoneData{n,1} >4) && (microphoneData{n,1}<=8))
                configureInput2.module           = slave1;
                configureInput2.microphoneData   = microphoneData;
                configureInput2.n                = n;

                connectedDevices.slave1...
                    = configureDT9837BChannels(configureInput2);
            end

            % Konfiguration der aktiven Kanäle (Slave2-Modul CH = 9:12)
            if ((microphoneData{n,1} >8) && (microphoneData{n,1}<=12))
                configureInput3.module           = slave2;
                configureInput3.microphoneData   = microphoneData;
                configureInput3.n                = n;

                connectedDevices.slave2...
                    = configureDT9837BChannels(configureInput3);
            end
        end

    case 4 % Drei DAQ-Module angeschlossen: Master-Slave-Slave-Slave
        try % Versuch der Anmeldung von drei Modulen
            displayGuiTextInfo(aInput.hInfoText,06);
            master = ...
                analoginput('dtol',...
                    hardwareInfo.InstalledBoardIds{masterIdx});
            slave1 = ...
```

```
407                 analoginput('dtol',...
408                     hardwareInfo.InstalledBoardIds{slaveIdx(1)});
409             slave2 = ...
410                 analoginput('dtol',...
411                     hardwareInfo.InstalledBoardIds{slaveIdx(2)});
412             slave3 = ...
413                 analoginput('dtol',...
414                     hardwareInfo.InstalledBoardIds{slaveIdx(3)});
415         catch % Falls das nicht funktioniert gibt es eine Fehlermeldung
416             displayGuiTextInfo(aInput.hInfoText,81);
417         end
418
419         userDateStr = datestr(now,30);
420
421         displayGuiTextInfo(aInput.hInfoText,07);
422         % Konfiguration der DAQ-Module-------------------------------------
423         set([master,slave1,slave2,slave3],...
424             'BufferingConfig'        , [blockLengthDAQ, lengthBuffer],...
425             'SamplesAcquiredFcnCount', blockLengthDAQ,...
426             'SamplesPerTrigger'      , Inf           ,...
427             'SampleRate'             , sampleRate    ,...
428             'LogToDiskMode'          , 'Index'       ,...
429             'LoggingMode'            , 'Disk'        ,...
430             'TimeOut'                , 2             ,...
431             'TriggerRepeat'          , 0             ,...
432             'TriggerType'            , 'Immediate'   );
433         % Kofiguration des Sync-Modus und der Callback-Funktion------------
434         set(master,...
435             'SyncMode'                  , 'Master' ,...
436             'SamplesAcquiredFcn'        , {@plotData1,aInput.hMamsGui},...
437             'LogFileName'               , ...
438             fullfile(folderStr,'DAQSession_Master',userDateStr));
439         set(slave1,...
440             'SyncMode'                  , 'Slave'       ,...
441             'SamplesAcquiredFcn'        , {@plotData2,aInput.hMamsGui},...
442             'LogFileName'               , ...
443             fullfile(folderStr,'DAQSession_Slave1',userDateStr));
444         set(slave2,...
445             'SyncMode'                  , 'Slave'       ,...
446             'SamplesAcquiredFcn'        , {@plotData3,aInput.hMamsGui},...
447             'LogFileName'               , ...
448             fullfile(folderStr,'DAQSession_Slave2',userDateStr));
449         set(slave3,...
450             'SyncMode'                  , 'Slave'       ,...
451             'SamplesAcquiredFcn'        , {@plotData4,aInput.hMamsGui},...
452             'LogFileName'               , ...
453             fullfile(folderStr,'DAQSession_Slave3',userDateStr));
454
455         % Konfiguration der aktiven Kanäle
456         for n= 1:numActiveChannels% Schleife über alle aktiven Kanäle
457             % Konfiguration der aktiven Kanäle (Master-Modul CH = 1:4)
458             if microphoneData{n,1} <= 4
459                 configureInput1.module           = master;
460                 configureInput1.microphoneData   = microphoneData;
461                 configureInput1.n                = n;
462
463                 connectedDevices.master =...
464                     configureDT9837BChannels(configureInput1);
465             end
466             % Konfiguration der aktiven Kanäle (Slave1-Modul CH = 5:8)
467             if ((microphoneData{n,1} >4) && (microphoneData{n,1}<=8))
```

```
                    configureInput2.module           = slave1;
                    configureInput2.microphoneData   = microphoneData;
                    configureInput2.n                = n;

                    connectedDevices.slave1...
                        = configureDT9837BChannels(configureInput2);
                end

                % Konfiguration der aktiven Kanäle (Slave2-Modul CH = 9:12)
                if ((microphoneData{n,1} >8) && (microphoneData{n,1}<=12))
                    configureInput3.module           = slave2;
                    configureInput3.microphoneData   = microphoneData;
                    configureInput3.n                = n;

                    connectedDevices.slave2...
                        = configureDT9837BChannels(configureInput3);
                end
                % Konfiguration der aktiven Kanäle (Slave3-Modul CH = 13:16)
                if ((microphoneData{n,1} >12) && (microphoneData{n,1}<=16))
                    configureInput4.module           = slave3;
                    configureInput4.microphoneData   = microphoneData;
                    configureInput4.n                = n;

                    connectedDevices.slave3...
                        = configureDT9837BChannels(configureInput4);
                end
            end
end
%-------------------------------------------------------------------------
%% Wegschreiben der Ergebnisse
names = fieldnames(connectedDevices);
if isempty(names)~=1
    out= connectedDevices;
end

out.blockLengthDAQ      = blockLengthDAQ; % Blocklänge [Samples]
out.folderStr           = folderStr; % Speicherort [String]
out.masterIdx           = masterIdx; % Index des angeschlossenen Slave Idx
out.microphoneData      = microphoneData; % Mikrofondaten [cell-struct]
out.moduleString        = moduleString; % Specher-Name [String]
out.numActiveChannels   = numActiveChannels; % Anzahl der aktiven Kanäle
% [double]
out.numDaqModules       = numDaqModules; % Anzahl der angeschlossenen
% DAQ-Module [double]
out.sampleRate          = sampleRate; % Abtastrate der DAQ-Module [double]
out.slaveIdx            = slaveIdx; % DAQ-Slave Index-Array [double array]

% aktualisierung des Anzeig Namen der GUI
set(aInput.hMamsGui,...
    'Name',[get(aInput.hMamsGui,'Name'),' fs=',num2str(sampleRate),...
    ' blockLength= ',num2str(blockLengthDAQ)]);
displayGuiTextInfo(aInput.hInfoText,08);
%-------------------------------------------------------------------------
end % Function end
```

```
function out = initFigureSetup(aInput)
%-------------------------------------------------------------------------
% Schnittstellenbeschreibung
%
```

```
5  % Autor:                Rick Plescher, 30.04.2015
6  %
7  % Funktionsaufruf:      out = initFigureSetup(aInput);
8  %
9  % Fkt.-beschreibung:    Die Funktion initiiert die Darstellungsfenster der
10 %                       aufzunehmenden Daten und das Menu der DAQ-Systems
11 %
12 % Eingabe:
13 %  aInput               : Eingabe-Struktur
14 %   .hMamsGui           : Handle der Haupt-GUI
15 %   .hInfoText          : Handle des Text-Info Fenster
16 %   .daqSetup           : Struktur mit DAQ-Setup-Daten (siehe initDaqSetup)
17 %
18 % Ausgabe:
19 %   out                 : Ausgabe-Struktur
20 %    .hAxes              : Handle der Achsen
21 %    .hLine              : Handle der Kurven
22 %    .hMenu              : Handle des Menüs
23 %    .hSubplot           : Handle des Subplots
24 %    .hPastSignalGui     : Handle der letzten aufgenommenen Daten (GUI)
25 %    .hPastSigPlot       : Handle der letzten aufgenommenen Daten (Kurve)
26 %    .pastSignalCounter  : Zähler der vergangenen Signalblöcke
27 %
28 % sontiges:
29 %   notwendigen Funktionen:
30 %       1. calibrateMic.m
31 %       2. close_callback.m
32 %       3. displayGuiMicInfo.m
33 %       4. displayGuiTextInfo.m
34 %       5. editMicrophoneData.m
35 %-------------------------------------------------------------------------
36 %% Intialisierung
37 hLine       = zeros(4,4); % Handle der Kurven der max. 16 DAQ-Kanäle
38 hAxes       = zeros(1,4); % Handle der Achsen für die 4 DAQ-Subplots
39 hSubplot    = zeros(1,6); % Handel der Subplot
40 fontSize    = 12; % Schriftgröße
41 yLimValue   = zeros(1,2);
42 yLimValue(1)= 20;
43 yLimValue(2)= 140;
44 %-------------------------------------------------------------------------
45 %% Programmcode
46 % Definition des x-Achse Tick/Label
47 xTickMat            = (1:2e3:aInput.daqSetup.blockLengthDAQ)-1;
48 xTickLabelCell      = cell(size(xTickMat));
49 xTickLabelCell(1)   = {'1'};
50 for n=2:numel(xTickMat)
51     xTickLabelCell(n) = {[num2str(fix(xTickMat(n)/1e3),'%i'),'k']};
52 end
53 % subplot 232: Master-Modul-------------------------------------------------
54 % Subplot initialisieren
55 hSubPlot(2) = subplot(2,3,2);
56 % 4 Kurven für den DAQ-Modul-Plot initialisieren
57 hLine(:,1)  = plot(zeros(aInput.daqSetup.blockLengthDAQ,4));
58 % setzen der Achsen
59 set(gca,...
60     'Parent'            , aInput.hMamsGui                   ,...
61     'XLim'              , [1,aInput.daqSetup.blockLengthDAQ],...
62     'XTick'             , xTickMat                          ,...
63     'XTickLabel'        , xTickLabelCell                    ,...
64     'YLim'              , [yLimValue(1),yLimValue(2)]);
65 % einlesen des Achsen-Handle
```

```
hAxes(1)      = gca;
% subplot 233: Slave1-Modul-----------------------------------------------
hSubPlot(3) = subplot(2,3,3);
hLine(:,2)   = plot(zeros(aInput.daqSetup.blockLengthDAQ,4));
set(gca,...
    'Parent'            , aInput.hMamsGui                   ,...
    'XLim'              , [1,aInput.daqSetup.blockLengthDAQ],...
    'XTick'             , xTickMat                          ,...
    'XTickLabel'        , xTickLabelCell                    ,...
    'YLim'              , [yLimValue(1),yLimValue(2)]);
hAxes(2)     = gca;
% subplot 235: Slave2-Modul-----------------------------------------------
hSubPlot(5) = subplot(2,3,5);
hLine(:,3)   = plot(zeros(aInput.daqSetup.blockLengthDAQ,4));
set(gca,...
    'Parent'            , aInput.hMamsGui                   ,...
    'XLim'              , [1,aInput.daqSetup.blockLengthDAQ],...
    'XTick'             , xTickMat                          ,...
    'XTickLabel'        , xTickLabelCell                    ,...
    'YLim'              , [yLimValue(1),yLimValue(2)]...
    );
hAxes(3)      = gca;
% subplot 236: Slave3-Modul-----------------------------------------------
hSubPlot(6) = subplot(2,3,6);
hLine(:,4)   = plot(zeros(aInput.daqSetup.blockLengthDAQ,4));
set(gca,...
    'Parent'            , aInput.hMamsGui                   ,...
    'XLim'              , [1,aInput.daqSetup.blockLengthDAQ],...
    'XTick'             , xTickMat                          ,...
    'XTickLabel'        , xTickLabelCell                    ,...
    'YLim'              , [yLimValue(1),yLimValue(2)]...
    );
hAxes(4)= gca;
%-------------------------------------------------------------------------
%% Signal-Past-Plot
% Diese Fenster zeigt bis zu einer gewissen Zeit an, was bisher aufgenommen
% wurde.

btnColor     = [1,1,1]; % Hintergrundfarbe des neuen Fenster
set(0,'Units','pixels'); % Fenster-Einheit auf Pixel setzten
screenSize   = get(0,'ScreenSize');  % einlesen der Bildschirm-Größe
figWidth     = screenSize(3)/2-50;   % Fensterbreite
figHeight    = screenSize(4)/4;      % Fensterhöhe
figPos       = ...                   % Fensterposition
    [20         , screenSize(4)-figHeight-50  ...
     figWidth   , figHeight];

hPastSignalGui  = figure(...
    'Units'             , 'pixels'          ,... % Einheit des Fenster
    'Color'             , btnColor          ,... % Hintergrundfarbe
    'Colormap'          , []                ,... %
    'HandleVisibility'  , 'on'              ,... % Zugang zum Handle
    'IntegerHandle'     , 'off'             ,... % Art der Zahl(für Handle)
    'MenuBar'           , 'none'            ,... % Menü
    'Name'              , ['MAMS - last 15 min overview '        ,...
                           '(maximum value of all channels from' ,...
                           ' the 1st DAQ-module/Master)'        ],...
    'NumberTitle'       , 'off'             ,... % Anzeige der Figure-Zahl
    'Position'          , figPos            ,... % Fenster-Position
    'Pointer'           , 'arrow'           ,... % Zeiger-Darstellung
    'Tag'               , 'pastSig-GUI'     ,... % Label für diese GUI
```

```
127     'UserData'          , []                ,... % spezifische Daten-Matrix
128     'Visible'           , 'on'              );   % Sichtbarkeit des GUI
129
130 maxDur              = 15;   % maximal Zeitdauer der darzustellenden letzten
131                             % Aufnahme-Minuten
132
133 % Anzahl der Blöcke um die maxDur Minuten darzustellen
134 numPastSigBlocks    = ...
135     ceil((maxDur*60*aInput.daqSetup.sampleRate)/...
136     aInput.daqSetup.blockLengthDAQ);
137 % Dauer eines DAQ-Blocks
138 dt                  = ...
139     aInput.daqSetup.blockLengthDAQ/aInput.daqSetup.sampleRate;
140 % Zeit-Vektor von -maxDur:0
141 t                   = fliplr((-1*((0:numPastSigBlocks-1)*dt)))';
142 % Initialisierung der Kurve für die bisherige Datenerfassung
143 hPastSigPlot        = plot(t,zeros(numPastSigBlocks,1));
144 % Initialisierung des Past-Signal-Zähler
145 pastSignalCounter   = 0;
146
147 title('signal overview','FontSize',26);
148 xlabel('last acquired max samples [min]');
149 ylabel('max signal level [dBSPL]');
150 set(gca,...
151     'XGrid'     , 'On'                                          ,...
152     'XLim'      , [-maxDur*60,0]                                ,...
153     'XTick'     , (-maxDur :2.5:0)*60                           ,...
154     'XTickLabel', {'-15';'-12.5';'-10';'-7.5';'-5';'-2.5';'0'}  ,...
155     'YGrid'     , 'On'                                          ,...
156     'YLim'      , [40,140]                                      ,...
157     'YTick'     , 40:20:140);
158 %-------------------------------------------------------------------------
159 %% Menu-Elemente initialisieren
160 displayGuiTextInfo(aInput.hInfoText,10);
161 % Initialisierung des Menüs-----------------------------------------------
162 hMenu(1) = uimenu(...
163     'Parent', aInput.hMamsGui               ,...
164     'Label', 'File');
165 hMenu(2) = uimenu(hMenu(1)                   ,...
166     'Accelerator'   , 'q'                   ,...
167     'Callback'      , 'close_callback(gcbf)',...
168     'Label'         , 'Close');
169 hMenu(3) = uimenu(...
170     'Label'         , 'Calibration'         ,...
171     'Parent'        , aInput.hMamsGui);
172 hMenu(4) = uimenu(...
173     'Accelerator'   ,'v'                    ,...
174     'Callback'      , 'displayGuiMicInfo'   ,...
175     'Label'         , 'view microphone data',...
176     'Parent'        , hMenu(3));
177 hMenu(5) = uimenu(...
178     'Accelerator'   , 'e'                   ,...
179     'Callback'      , 'editMicrophoneData'  ,...
180     'Label'         , 'edit microphone data',...
181     'Parent'        , hMenu(3));
182 hMenu(6) = uimenu(...
183     'Accelerator'   ,'c'                    ,...
184     'Callback'      , 'calibrateMic(gcbf)'  ,...
185     'Label'         , 'calibrate Microphone',...
186     'Parent'        , hMenu(3));
187
```

```
188 %% Darstellung der aktiven Modul-Fenster
189 % Unterscheidung nach Anzahl der DAQ-Module
190 switch aInput.daqSetup.numDaqModules
191     case 1
192         % Aktivieren des Master-Modul Fenster-----------------------------
193         set(hAxes(1),...
194             'FontSize'  , fontSize,...
195             'XGrid'     , 'On',...
196             'YGrid'     , 'On');
197         xlabel(hAxes(1) , '\rightarrow samples [-]');
198         ylabel(hAxes(1) , '\rightarrow sound level [dBSPL]');
199         title(hAxes(1)  , 'Master (Ch01-Ch04)');
200         % Deaktivieren des übrigen Fenster--------------------------------
201         set(hLine(:,2)  , 'Visible','Off');
202         set(hSubPlot(3) , 'Visible','Off');
203         set(hLine(:,3)  , 'Visible','Off');
204         set(hSubPlot(5) , 'Visible','Off');
205         set(hLine(:,4)  , 'Visible','Off');
206         set(hSubPlot(6) , 'Visible','Off');
207     case 2
208         % Aktivieren des Master-Modul Fenster-----------------------------
209         set(hAxes(1),...
210             'FontSize'  , fontSize,...
211             'XGrid'     , 'On',...
212             'YGrid'     , 'On');
213         xlabel(hAxes(1) , '\rightarrow samples [-]');
214         ylabel(hAxes(1) , '\rightarrow sound level [dBSPL]');
215         title(hAxes(1)  , 'Master (Ch01-Ch04)');
216         % Aktivieren des Slave1-Modul Fenster-----------------------------
217         set(hAxes(2),...
218             'FontSize'  , fontSize,...
219             'XGrid'     , 'On',...
220             'YGrid'     , 'On');
221         xlabel(hAxes(2) , '\rightarrow samples [-]');
222         ylabel(hAxes(2) , '\rightarrow sound level [dBSPL]');
223         title(hAxes(2)  , 'Slave1 (Ch05-Ch08)');
224         % Deaktivieren des übrigen Fenster--------------------------------
225         set(hLine(:,3)  , 'Visible','Off');
226         set(hSubPlot(5) , 'Visible','Off');
227         set(hLine(:,4)  , 'Visible','Off');
228         set(hSubPlot(6) , 'Visible','Off');
229     case 3
230         % Aktivieren des Master-Modul Fenster-----------------------------
231         set(hAxes(1),...
232             'FontSize'  , fontSize,...
233             'XGrid'     , 'On',...
234             'YGrid'     , 'On');
235         xlabel(hAxes(1) , '\rightarrow samples [-]');
236         ylabel(hAxes(1) , '\rightarrow sound level [dBSPL]');
237         title(hAxes(1)  , 'Master (Ch01-Ch04)');
238         % Aktivieren des Slave1-Modul Fenster-----------------------------
239         set(hAxes(2),...
240             'FontSize'  , fontSize,...
241             'XGrid'     , 'On',...
242             'YGrid'     , 'On');
243         xlabel(hAxes(2) , '\rightarrow samples [-]');
244         ylabel(hAxes(2) , '\rightarrow sound level [dBSPL]');
245         title(hAxes(2)  , 'Slave1 (Ch05-Ch08)');
246         % Aktivieren des Slave2-Modul Fenster-----------------------------
247         set(hAxes(3),...
248             'FontSize'  , fontSize,...
```

```
249             'XGrid'      , 'On',...
250             'YGrid'      , 'On');
251         xlabel(hAxes(3) , '\rightarrow samples [-]');
252         ylabel(hAxes(3) , '\rightarrow sound level [dBSPL]');
253         title(hAxes(3)  , 'Slave2 (Ch09-Ch12)');
254         % Deaktivieren des übrigen Fenster--------------------------------
255         set(hLine(:,4)  , 'Visible','Off');
256         set(hSubPlot(6) , 'Visible','Off');
257     case 4
258         % Aktivieren des Master-Modul Fenster-----------------------------
259         set(hAxes(1),...
260             'FontSize'  , fontSize,...
261             'XGrid'      , 'On',...
262             'YGrid'      , 'On');
263         xlabel(hAxes(1) , '\rightarrow samples [-]');
264         ylabel(hAxes(1) , '\rightarrow sound level [dBSPL]');
265         title(hAxes(1)  , 'Master (Ch01-Ch04)');
266         % Aktivieren des Slave1-Modul Fenster-----------------------------
267         set(hAxes(2),...
268             'FontSize'  , fontSize,...
269             'XGrid'      , 'On',...
270             'YGrid'      , 'On');
271         xlabel(hAxes(2) , '\rightarrow samples [-]');
272         ylabel(hAxes(2) , '\rightarrow sound level [dBSPL]');
273         title(hAxes(2)  , 'Slave1 (Ch05-Ch08)');
274         % Aktivieren des Slave2-Modul Fenster-----------------------------
275         set(hAxes(3),...
276             'FontSize'  , fontSize,...
277             'XGrid'      , 'On',...
278             'YGrid'      , 'On');
279         xlabel(hAxes(3) , '\rightarrow samples [-]');
280         ylabel(hAxes(3) , '\rightarrow sound level [dBSPL]');
281         title(hAxes(3)  , 'Slave2 (Ch09-Ch12)');
282         % Aktivieren des Slave3-Modul Fenster-----------------------------
283         set(hAxes(4),...
284             'FontSize'  , fontSize,...
285             'XGrid'      , 'On',...
286             'YGrid'      , 'On');
287         xlabel(hAxes(4) , '\rightarrow samples [-]');
288         ylabel(hAxes(4) , '\rightarrow sound level [dBSPL]');
289         title(hAxes(4)  , 'Slave3 (Ch13-Ch16)');
290 end
291 %-----------------------------------------------------------------------
292 %% Wegschreiben der Ergebnisse
293 out.hAxes               = hAxes;
294 out.hLine                = hLine;
295 out.hMenu                = hMenu;
296 out.hSubplot             = hSubplot;
297 out.hPastSignalGui       = hPastSignalGui;
298 out.hPastSigPlot         = hPastSigPlot;
299 out.pastSignalCounter    = pastSignalCounter;
300 %-----------------------------------------------------------------------
301 end
```

```
1 function keyEvent_callback(gcobject)
2 %-----------------------------------------------------------------------
3 % Schnittstellenbeschreibung
4 %
5 % Autor:                Rick Plescher, 30.04.2015
```

```
6  %
7  % Funktionsaufruf:      startStop_callback(gcobject)
8  %
9  % Fkt.-beschreibung:    Die Funktion wird beim Drücken des
10 %                       Start/Stop-Button in der Main-GUI ausgeführt. Sie
11 %                       steuert im Wesentlichen den Ablauf der Messung.
12 %
13 % Eingabe:
14 %   gcobject:           handle der Figures, welches diese Funktion
15 %                       aufgerufen hat.
16 %
17 % Ausgabe:              - keine Ausgabe -
18 %
19 %
20 % sontiges:
21 %-------------------------------------------------------------------------
22 %% Intialisierung
23 % Einlesen objekt-spezifischen Daten des MAMS-GUI
24 userData    = get(gcobject,'UserData');
25 % Einlesen der letzten vom Benutzer eingegebenen Taste
26 eventKey    = get(gcobject,'CurrentCharacter');
27 %-------------------------------------------------------------------------
28 %% Programmcode
29 % Abfrage welchen Status die Datenerfassung gerade hat und ob mit einem
30 % Druck auf die Leertaste ' ' der Status geändert werden soll.
31 % userData.daqState = 'Start':= Die Datenerfassung ist bereit zu aufnehmen,
32 %                               ist aber nicht aktiv.
33 % userData.daqState = 'Stop' := Die Datenerfassung ist aktiv und wird mit
34 %                               einem Druck auf die Leertaste beendet
35
36 if (strcmp(userData.daqState,'Start')&& strcmp(eventKey,' '));
37     userData.daqState = 'Stop'; % Status-Wechsel
38     set(gcobject,'Color',[0.8,1,0.8]); % neue Hintergrundfrabe der MAMS-GUI
39     % Initialisieren des Plot für die letzten Minuten der
40     % Signalaufzeichnung
41     resetOverallSignal = get(userData.hPastSigPlot,'YData');
42     set(userData.hPastSigPlot,...
43         'YData' , zeros(size(resetOverallSignal)));
44
45     % Einlesen der Mikrofonedaten+++++++++++++++++++++++++++++++++++++++++
46     readMicInput.file       = fullfile(cd,'microphoneData.mat');
47     readMicInput.header     = true;
48     userData.microphoneData = readMicrophoneData(readMicInput);
49     %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
50
51     % Erzeugung des Datums-String für die Log-Files
52     userData.dateStr    = datestr(now,30);
53
54     % Switch-Abfrage für die Unterscheidung der ANzahl der
55     % angeschlossenenen Module
56     switch userData.numDaqModules
57         case 1 % Ein Modul angeschlossen: Master
58             % Einstellen des Sync-Modus des Master-Module
59             set(userData.master, 'SyncMode' , 'Master');
60             % Einstellen des Log-FileName und der Callback-Funktion
61             set(userData.master,...
62                 'LogFileName'               , ...
63                     fullfile(userData.folderStr,...
64                     'DAQSession_Master',userData.dateStr),...
65                 'SamplesAcquiredFcn'        , {@plotData1,gcobject});
66             % Aktualisierung des neuen GUI-Name
```

```
67              set(gcobject,...
68                  'Name', ...
69                  ['Graphical User Interface - ',...
70                   'Mobile Array Measurement System (MAMS)-',...
71                   userData.dateStr]);
72              % Aktualisierung des neuen Speichernamen und Speichern der
73              % Mikrofondaten
74              saveMicDataInput.saveFileStr=...
75                  fullfile(userData.folderStr,...
76                      'DAQSession_MicrophoneData',[userData.dateStr,'.mat']);
77              saveMicDataInput.header     = true; % Speichern mit Header
78              saveDAQMicrophoneData(saveMicDataInput);
79              %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
80              % Start des DAQ-Modul++++++++++++++++++++++++++++++++++++++++
81              start(userData.master);
82              %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
83              % Auslesen der relevanten DAQ-Parameter
84              tmp.dateStr         = userData.dateStr;
85              tmp.eventKey        = eventKey;
86              tmp.blockLengthDAQ  = userData.blockLengthDAQ;
87              tmp.sampleRate      = userData.sampleRate;
88              set(userData.hInfoText,'Userdata',tmp);
89              % Aktualisierung des DAQ-Infofenster in der GUI
90              displayGuiTextInfo(userData.hInfoText,12);
91          case 2 % Zwei Module angeschlossen: Master-Slave1
92              % Einstellen des Sync-Modus der DAQ-Module
93              set(userData.master, 'SyncMode' , 'Master');
94              set(userData.slave1, 'SyncMode' , 'Slave');
95              %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
96              % Einstellen des Log-FileName und der Callback-Funktion
97              set(userData.slave1,...
98                  'LogFileName'               , ...
99                      fullfile(userData.folderStr,...
100                     'DAQSession_Slave1',userData.dateStr),...
101                 'SamplesAcquiredFcn'        , {@plotData2,gcobject});
102             set(userData.master,...
103                 'LogFileName'               , ...
104                     fullfile(userData.folderStr,...
105                     'DAQSession_Master',userData.dateStr),...
106                 'SamplesAcquiredFcn'        , {@plotData1,gcobject});
107             %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
108             % Aktualisierung des neuen GUI-Name
109             set(gcobject,...
110                 'Name',['Graphical User Interface -',...
111                 ' Mobile Array Measurement System (MAMS)-',...
112                 userData.dateStr]);
113             %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
114             % Aktualisierung des neuen Speichernamen und Speichern der
115             % Mikrofondaten
116             saveMicDataInput.saveFileStr=...
117                 fullfile(userData.folderStr,...
118                 'DAQSession_MicrophoneData',[userData.dateStr,'.mat']);
119             saveMicDataInput.header     = true; % Speichern mit Header
120             saveDAQMicrophoneData(saveMicDataInput);
121             %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
122             % Start der DAQ-Module++++++++++++++++++++++++++++++++++++++
123             start(userData.slave1);
124             start(userData.master);
125             %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
126             % Auslesen der relevanten DAQ-Parameter
127             tmp.dateStr         = userData.dateStr;
```

```
            tmp.eventKey        = eventKey;
            tmp.blockLengthDAQ  = userData.blockLengthDAQ;
            tmp.sampleRate      = userData.sampleRate;

            set(userData.hInfoText,'Userdata',tmp);
            % Aktualisierung des DAQ-Infofenster in der GUI
            displayGuiTextInfo(userData.hInfoText,12);
        case 3 % Drei Module angeschlossen: Master-Slave1-Slave2
            % Einstellen des Sync-Modus der DAQ-Module
            set(userData.master, 'SyncMode' , 'Master');
            set(userData.slave1, 'SyncMode' , 'Slave');
            set(userData.slave2, 'SyncMode' , 'Slave');
            %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            % Einstellen des Log-FileName und der Callback-Funktion
            set(userData.slave2,...
                'LogFileName'               , ...
                    fullfile(userData.folderStr,...
                    'DAQSession_Slave2',userData.dateStr),...
                'SamplesAcquiredFcn'        , {@plotData3,gcobject});
            set(userData.slave1,...
                'LogFileName'               , ...
                    fullfile(userData.folderStr,...
                    'DAQSession_Slave1',userData.dateStr),...
                'SamplesAcquiredFcn'        , {@plotData2,gcobject});
            set(userData.master,...
                'LogFileName'               , ...
                    fullfile(userData.folderStr,...
                    'DAQSession_Master',userData.dateStr),...
                'SamplesAcquiredFcn'        , {@plotData1,gcobject});
            %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            % Aktualisierung des neuen GUI-Name
            set(gcobject,...
                'Name',['Graphical User Interface -',...
                ' Mobile Array Measurement System (MAMS)-',...
                userData.dateStr]);
            %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            % Aktualisierung des neuen Speichernamen und Speichern der
            % Mikrofondaten
            saveMicDataInput.saveFileStr=...
                fullfile(userData.folderStr,...
                'DAQSession_MicrophoneData',[userData.dateStr,'.mat']);
            saveMicDataInput.header     = true; % Speichern mit Header
            saveDAQMicrophoneData(saveMicDataInput);
            %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            % Start der DAQ-Module++++++++++++++++++++++++++++++++++++++++
            start(userData.slave2);
            start(userData.slave1);
            start(userData.master);
            %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            % Auslesen der relevanten DAQ-Parameter
            tmp.dateStr         = userData.dateStr;
            tmp.eventKey        = eventKey;
            tmp.blockLengthDAQ  = userData.blockLengthDAQ;
            tmp.sampleRate      = userData.sampleRate;
            set(userData.hInfoText,'Userdata',tmp);
            % Aktualisierung des DAQ-Infofenster in der GUI
            displayGuiTextInfo(userData.hInfoText,12);
        case 4 % Vier Module angeschlossen: Master-Slave1-Slave2-Slave3
            % Einstellen des Sync-Modus der DAQ-Module
            set(userData.master, 'SyncMode' , 'Master');
            set(userData.slave1, 'SyncMode' , 'Slave');
```

```
189             set(userData.slave2, 'SyncMode' , 'Slave');
190             set(userData.slave3, 'SyncMode' , 'Slave');
191             %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
192             % Einstellen des Log-FileName und der Callback-Funktion
193             set(userData.slave3,...
194                 'LogFileName'               , ...
195                     fullfile(userData.folderStr,...
196                     'DAQSession_Slave3',userData.dateStr),...
197                 'SamplesAcquiredFcn'        , {@plotData4,gcobject});
198             set(userData.slave2,...
199                 'LogFileName'               , ...
200                     fullfile(userData.folderStr,...
201                     'DAQSession_Slave2',userData.dateStr),...
202                 'SamplesAcquiredFcn'        , {@plotData3,gcobject});
203             set(userData.slave1,...
204                 'LogFileName'               , ...
205                     fullfile(userData.folderStr,...
206                     'DAQSession_Slave1',userData.dateStr),...
207                 'SamplesAcquiredFcn'        , {@plotData2,gcobject});
208             set(userData.master,...
209                 'LogFileName'               , ...
210                     fullfile(userData.folderStr,...
211                     'DAQSession_Master',userData.dateStr),...
212                 'SamplesAcquiredFcn'        , {@plotData1,gcobject});
213             %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
214             % Aktualisierung des neuen GUI-Name
215             set(gcobject,...
216                 'Name',['Graphical User Interface -',...
217                 ' Mobile Array Measurement System (MAMS)-',...
218                 userData.dateStr]);
219             %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
220             % Aktualisierung des neuen Speichernamen und Speichern der
221             % Mikrofondaten
222             saveMicDataInput.saveFileStr=...
223                 fullfile(userData.folderStr,...
224                 'DAQSession_MicrophoneData',[userData.dateStr,'.mat']);
225             saveMicDataInput.header     = true;
226             saveDAQMicrophoneData(saveMicDataInput);
227             %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
228             % Start der DAQ-Module++++++++++++++++++++++++++++++++++++++++++
229             start(userData.slave3);
230             start(userData.slave2);
231             start(userData.slave1);
232             start(userData.master);
233             %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
234             % Auslesen der relevanten DAQ-Parameter
235             tmp.dateStr         = userData.dateStr;
236             tmp.eventKey        = eventKey;
237             tmp.blockLengthDAQ  = userData.blockLengthDAQ;
238             tmp.sampleRate      = userData.sampleRate;
239             set(userData.hInfoText,'Userdata',tmp);
240             % Aktualisierung des DAQ-Infofenster in der GUI
241             displayGuiTextInfo(userData.hInfoText,12);
242     end
243
244     % Wenn der DAQ-Statuts auf Stop steht und die Leertaste gedrückt wurde
245     % werden die laufenden Module gestoppt
246 elseif (strcmp(userData.daqState,'Stop')&& strcmp(eventKey,' '))
247     % Aktualisierung der Hintergrundfarbe
248     set(gcobject,'Color',[1,0.8,0.8]);
249     userData.daqState = 'Start'; % Wechsel des DAQ Status auf Bereitschaft
```

```
                                    % für eine weitere DAQ-Session
    % Switch-Abfrage zum Stoppen der DAQ-Module in Abhängikeit der Anzahl
    % der bereits angemeldeten Module
    % Das Master-Module wird immer als erstes gestoppt
    switch userData.numDaqModules
        case 1
            stop(userData.master);
        case 2
            stop(userData.master);
            stop(userData.slave1);
        case 3
            stop(userData.master);
            stop(userData.slave1);
            stop(userData.slave2);
        case 4
            stop(userData.master);
            stop(userData.slave1);
            stop(userData.slave2);
            stop(userData.slave3);
    end
    % Aktualisierung des DAQ-Infofenster
    displayGuiTextInfo(userData.hInfoText,11);
    % Zurücksetzen des Zähler des bisherigen Signalverlaufs
    userData.pastSignalCounter = 0;
end

% Wenn die Datenerfassung noch aktive ist und ein Key-Evvent erfolgt wird
% die Tastatureingabe aufgezeichnet
if (strcmp(userData.daqState,'Stop'))
    try % Prüfen ob bereits ein Event-Log angelegt wurde
        tmp     = load(...
            fullfile(...
                userData.folderStr,'DAQSession_LogData',...
                [userData.dateStr,'.mat']),'eventLog');
        eventLog= tmp.eventLog;
    catch % Wenn kein Event-Log angelegt wurde, wird eines jetzt mit Header
          % angelegt
        eventLog = {'time Stamp (UTC)', 'event Number'};
        save(fullfile(userData.folderStr,'DAQSession_LogData',...
            [userData.dateStr,'.mat']),'eventLog');
    end
    % Alle Tastatureingaben außßer der Leertaste werden als Event-Log
    % Eintrag aufgezeichnet
    % Platzhalter: An dieser Stelle ist eine weitere Einschränkung der
    % zulässigen Event-Key Log Bereichs durch die Abfrage möglich z.B. nur
    % a-z und/oder A-Z und/oder 0-9 ...
    if (strcmp(eventKey,' ')==0)
        % Erstellung des neuen Event-Key Stempel
        tmp     = {datestr(now,30),['Event ''',eventKey,''''']};
        % Kaskadierung des neuen mit den alten Stemple
        eventLog= [eventLog;tmp];

        % Erstellung eines String für die MAMS-GUI
        eventString = {[tmp{2},' logged at '];tmp{1};' (UTC)'};
        % Wegschreiben der Ergebnisse in die MAMS-GUI
        displayUserData.dateStr         = userData.dateStr;
        displayUserData.eventString     = eventString;
        displayUserData.blockLengthDAQ  = userData.blockLengthDAQ;
        displayUserData.sampleRate      = userData.sampleRate;
        % Aktualiseirung des Info-Fenster
        set(userData.hInfoText,'Userdata',displayUserData);
```

```
311         displayGuiTextInfo(userData.hInfoText,13);
312         % Speichern der aktuellen Event - Log-Datei
313         save(fullfile(userData.folderStr,...
314             'DAQSession_LogData',[userData.dateStr,'.mat']),'eventLog');
315     end
316 end
317 %-------------------------------------------------------------------------
318 %% Wegschreiben der Ergebnisse
319 set(gcobject,'UserData', userData);
320 %-------------------------------------------------------------------------
321 end
```

```
1  function loadPresetMicData_mouseCallback(~,~,aTableHandle)
2  %-------------------------------------------------------------------------
3  % Schnittstellenbeschreibung
4  %
5  % Autor:                Rick Plescher, 30.04.2015
6  %
7  % Funktionsaufruf:      loadPresetMicData_mouseCallback(~,~,aTableHandle)
8  %
9  % Fkt.-beschreibung:    Lädt auf Mausklick ein Preset (nur auf den Button)
10 %
11 % Eingabe:
12 %   aTable              Handle ur MicDataTabelle
13 %
14 % Ausgabe:              indrekt übner neue microphoneData.mat
15 %
16 %
17 % sontiges:
18 %   notwendige Funktion: readMicrophoneData
19 %-------------------------------------------------------------------------
20 %% Programmcode
21 [fileName,pathName] = uigetfile(...
22     '.mat',...
23     'Please choose a microphone setup file',...
24     'MultiSelect'    , 'Off');
25
26 if fileName ~=0
27     readMicInput.file   = fullfile([pathName,fileName]);
28     readMicInput.header = true;
29     microphoneData      = readMicrophoneData(readMicInput);
30
31     set(aTableHandle,'Data',microphoneData(2:end,4:end));
32
33     save(fullfile(cd,'microphoneData.mat'),'microphoneData');
34 else
35     warndlg('No new Mic-Preset loaded!');
36 end
37 %-------------------------------------------------------------------------
38 %% Wegschreiben der Ergebnisse
39
40 %-------------------------------------------------------------------------
41 end
```

```
1  function loadSavePresetMicData_keyCallback(obj,~,aTableHandle)
2  %-------------------------------------------------------------------------
3  % Schnittstellenbeschreibung
4  %
```

```
5  % Autor:                Rick Plescher, 30.04.2015
6  %
7  % Funktionsaufruf:      loadSavePresetMicData_keyCallback
8  %
9  % Fkt.-beschreibung:    Speichert/lädt die Mikrofoneinstellungen
10 %
11 % Eingabe:
12 % aTabelHandle:         Handle zu Tabelle der editMicrphoneData()-funktion
13 %
14 % Ausgabe:              keine direkte
15 %                       indirekt wird eine neues Setup gespeichert7geladen.
16 %
17 %
18 % sontiges:
19 %   notwendige Funktion: readMicrophoneData
20 %-------------------------------------------------------------------------
21
22 %% Intialisierung
23
24 %-------------------------------------------------------------------------
25 %% Überprüfung der Eingabe-Parameter
26
27 %-------------------------------------------------------------------------
28 %% Programmcode
29 if (strcmp(get(obj,'CurrentCharacter'),'l'))
30     [fileName,pathName] = uigetfile(...
31         '.mat',...
32         'Please choose a microphone setup file',...
33         'MultiSelect'    , 'Off');
34
35     readMicInput.file   = fullfile([pathName,fileName]);
36     readMicInput.header = true;
37     microphoneData      = readMicrophoneData(readMicInput);
38
39     set(aTableHandle,'Data',microphoneData(2:end,4:end));
40 end
41
42 if (strcmp(get(obj,'CurrentCharacter'),'s'))
43     [fileName,pathName] = uiputfile(...
44         '.mat',...
45         'Please save your new microphone setup file');
46
47     microphoneItems = { ...
48         'Num'           ,'Ch-Num'       ,'HwCh-Num' ,'Model',...
49         'Serialnumber'  ,'Sensitivity'  ,'Last Calibration'};
50
51     data            = get(aTableHandle,'Data');
52     tmp             = cell(size(data,1),7);
53     for n = 1:size(data,1)
54         tmp(n,:)            = ..
55         {n,n,mod(n-1,4),data{n,1},data{n,2},data{n,3},data{n,4}};
56     end
57     microphoneData  = [microphoneItems;tmp];
58     save([pathName,fileName],'microphoneData');
59 end
60 %-------------------------------------------------------------------------
61 %% Wegschreiben der Ergebnisse
62
63 %-------------------------------------------------------------------------
64 end
```

```
%-------------------------------------------------------------------------
clc; % Löschen de Einträge des 'Command Window'
daqreset; % entfernt alle geladene DLL, Mex-Files und DAQ-Objekte
clear all; % Löscht alle Variablen aus den Arbeitsspeicher
close all; % Schließt alle Fenster
%-------------------------------------------------------------------------
%% Initialisierung der Haupt-GUI
set(0,'Units','pixels'); % Einheit der Main-GUI in Pixel setzen
screenUnits = get(0,'Units'); % Speichern der Bildschirmeinheit
screenSize  = get(0,'ScreenSize'); % Abfrage der Bildschirmgröße

figWidth    = screenSize(3)/2; % Definition der Fensterbreite
figHeight   = screenSize(4)-100; % Definition der Fensterhöhe
figPos      = ...                % Definition der Fensterposition
    [screenSize(3)/2, 40  ...
     figWidth       , figHeight];

 btnColor   = [1,1,1]; % Definition der Hintergrundfarbe der Main-GUI

hMamsGui    = figure(... % Definition des Main-GUI Fensters
    'Units'             , 'pixels'                        ,...
    'Color'             , btnColor                        ,...
    'Colormap'          , []                              ,...
    'DeleteFcn'         , 'close_callback(gcbf)'          ,...
    'HandleVisibility'  , 'on'                            ,...
    'KeyPressFcn'       , 'keyEvent_callback(gcbf)'       ,...
    'MenuBar'           , 'none'                          ,...
    'Name'              , ['Graphical User Interface -'   ,...
    ' Mobile Array Measurement System (MAMS)']            ,...
    'NumberTitle'       , 'off'                           ,...
    'Pointer'           , 'arrow'                         ,...
    'Position'          , figPos                          ,...
    'Tag'               , 'Main-GUI'                      ,...
    'UserData'          , []                              ,...
    'Visible'           , 'On'                            );
% subplot 231: Info Fenster-----------------------------------------------
hInfo       = subplot(2,3,1); % Subplot-Fenster erzeugen
hInfoAxe    = gca; % aktuelles Handle laden
hInfoText   = text(-0.6,0.8,'','FontSize',12); % Text-Objekt erzeugen

set(hInfo,'Visible','Off'); % Sichtbarkeit des Fenster ausstellen
set(hInfoAxe,'Visible','Off'); % Sichtbarkeit der Achse ausstellen

displayGuiTextInfo(hInfoText,01); % Ausgabe eines Info-Nachricht
%-------------------------------------------------------------------------
%% Auswahl des Mikrofon-Setup
editMicrophoneData() % Benutzer nach Mikrofon-Setup fragen
%-------------------------------------------------------------------------
%% Intialisierung der Module
% Definition der Eingabe-Struktur zur Intialisierung der DAQ-Module
daqSetupInput.hInfoText     = hInfoText;
daqSetupInput.hMamsGui      = hMamsGui;
% Funktionsaufruf zur Intialisierung der DAQ-Module
daqSetup                    = initDaqSetup(daqSetupInput);
%-------------------------------------------------------------------------
%% Initialisierung der Subplots
% Definition der Eingabe-Struktur zur Intialisierung der Fenster
figureSetupInput.hMamsGui   = hMamsGui;
figureSetupInput.hInfoText  = hInfoText;
figureSetupInput.daqSetup   = daqSetup;
```

```
% Funktionsaufruf zur initialisierung der GUI-Fenster
figSetup                    = initFigureSetup(figureSetupInput);
%-------------------------------------------------------------------------
%% Definition des UserData-Struktur
% Die Userdata-Struktur der Main-GUI enthält alle Informationen und
% Daten während einer laufenden Messungen.

% Daten die in dieser Datei erzeugt wurden--------------------------------
userData.hInfo              = hInfo;    % Handle des Info-Fenster
userData.hInfoAxe           = hInfoAxe; % Handle der Achsen
userData.hInfoText          = hInfoText;% Handle des Info-Text-Anzeige
%-------------------------------------------------------------------------
% Daten die in der initDaqSetup-Dunktion erzeugt wurden--------------------
userData.blockLengthDAQ     = daqSetup.blockLengthDAQ;
% Dateipfad der zu speichernden Datei
userData.folderStr          = daqSetup.folderStr;
% Index des Modules, welches als Master definiert wurde
userData.masterIdx          = daqSetup.masterIdx;
% Zell-Struktur mit Mikrofon-Daten
userData.microphoneData     = daqSetup.microphoneData;
% Namen der DAQ-Module
userData.moduleString       = daqSetup.moduleString;
% Anzahl der aktive Kanäle
userData.numActiveChannels  = daqSetup.numActiveChannels;
% Anzahl der DAQ-Module
userData.numDaqModules      = daqSetup.numDaqModules;
% Abtastrate der DAQ-Module
userData.sampleRate         = daqSetup.sampleRate;
% Vektor mit den Indizes der Slave-DAQ-Module
userData.slaveIdx           = daqSetup.slaveIdx;
% Auswahl des angeschlossenen DAQ-Module
switch userData.numDaqModules
    case 1
        userData.master     = daqSetup.master;
    case 2
        userData.master     = daqSetup.master;
        userData.slave1     = daqSetup.slave1;
    case 3
        userData.master     = daqSetup.master;
        userData.slave1     = daqSetup.slave1;
        userData.slave2     = daqSetup.slave2;
    case 4
        userData.master     = daqSetup.master;
        userData.slave1     = daqSetup.slave1;
        userData.slave2     = daqSetup.slave2;
        userData.slave3     = daqSetup.slave3;
end
%-------------------------------------------------------------------------
% Daten die in der initFigureSetup erzeugt wurden--------------------------
% Handle der Subplot-Achsen
userData.hAxes              = figSetup.hAxes;
% Handle der Signal-Pegel-Darstellung
userData.hLine              = figSetup.hLine;
% Handle des Main-Gui-Menus
userData.hMenu              = figSetup.hMenu;
% Handle der Subplots
userData.hSubplot           = figSetup.hSubplot;
% Handle der Past-Signal-Kurve
userData.hPastSigPlot       = figSetup.hPastSigPlot;
% Handle Past-Signal-Overview
userData.hPastSignalGui     = figSetup.hPastSignalGui;
```

```
122 % Schleifenzähler der Past-Signal-Kurve
123 userData.pastSignalCounter  = figSetup.pastSignalCounter;
124 %-------------------------------------------------------------------------
125 % Die Datenerfassung messbereit einstellen
126 userData.daqState           = 'Start';
127 % Aktualisieren der neuen Daten des UserData-Matrix in der Main-GUI
128 set(hMamsGui,'UserData',userData);
129 % Ausgabe der Statusmeldung: Bereit zur Messung!
130 displayGuiTextInfo(userData.hInfoText,11);
131 %-------------------------------------------------------------------------
132 %%
133 helpdlg('MAMS is now ready to start!','');
134 %-------------------------------------------------------------------------
```

```
1  function plotData1(aObject,~, gcobject)
2  %-------------------------------------------------------------------------
3  % Schnittstellenbeschreibung
4  %
5  % Autor:                Rick Plescher, 30.04.2015
6  %
7  % Funktionsaufruf:      plotData1(obj,~, gcobject)
8  %
9  % Fkt.-beschreibung:    Diese Funktion wird immer ausgeführt wenn der
10 %                       SamplesAcquiredFcnCount des Master-Modul erreicht
11 %                       wird. Es wird das Fenster mit den Signalen des
12 %                       Master-Modul aktualisiert und in dieser Funktion
13 %                       zusätzlich das Fenster mit dem bisherigen
14 %                       Signalverlauf aktualsiert wird
15 %
16 % Eingabe:
17 %  aObject              : Objekt-Handle zu dem Master-Module mit seinen
18 %                         aktuellen Daten und EInstellungen
19 %
20 % Ausgabe:              keine direkte Ausgabe
21 %
22 %
23 % sontiges:
24 %                       Zum korrekten Ablauf ist die Function calcSPl.m
25 %                       notwendig. Außerdem muss das Fenster des bisherigen
26 %                       Signalverlaufs geöffnet sein.
27 %-------------------------------------------------------------------------
28 %% Programmcode
29 % Einlesen der aktuellen Daten
30 userData = get(gcobject,'UserData');
31 % Einlesen der aktuellen Mikrofondaten
32 readMicInput.file       = fullfile(fullfile(cd,'microphoneData.mat'));
33 readMicInput.header     = false;
34 userData.microphoneData = readMicrophoneData(readMicInput);
35 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
36 % Abfrage ob das Master-Module immer noch aktiv ist
37 if isrunning(aObject)
38     data                = ... % Vorrausschau der aktuellen erfassten Daten
39         peekdata(userData.master, userData.master.SamplesAcquiredFcnCount);
40     numActivechannels   = ... % Ermittlung der aktuell angemeldeten Kanäle
41         size(get(get(userData.master,'Channel'),'HwChannel'),1);
42
43     % Schleife zur Berechnung des pegelrichtige Zeitsignals
44     for n = 1:numActivechannels % Schleife maximal 1 bis 4
45         calcSplInput.signal     = data(:,n);
```

```
46          calcSplInput.sensitivity= userData.microphoneData{n,6};
47          set(userData.hLine(n,1) , 'YData',calcSpl(calcSplInput));
48      end
49
50      % Darstellung des bisherigen Signalverlauf (jeweils das Maximum
51      % des Master-Modul)
52      if (userData.pastSignalCounter < ...
53              length(get(userData.hPastSigPlot,'YData')))
54          % Inkrementieren des Signalverlauf-Zähler
55          userData.pastSignalCounter      = ...
56              userData.pastSignalCounter+1;
57          % Ermittlung des größten Amplitudenwertes aller Kanäle des
58          % Master-Moduls
59          tmpMax                          =...
60              max(max(get(userData.hLine(1,1),'YData')));
61          % Einlesen des bisherigen Signalverlaufs
62          tmp                             =...
63              get(userData.hPastSigPlot,'YData');
64          % Setzen des neuen Wertes für den bisherigen Signalverlauf
65          tmp(userData.pastSignalCounter) = tmpMax;
66          % Aktualisierung des bisherigen Signalverlauf
67          set(userData.hPastSigPlot,'YData',tmp);
68      else % Wenn der Zähler größer ist als die Länge des darzustellenden
69           % Signalverlaufs werden nur die letzten aktuellen Daten
70           % dargestellt. DEFAULT-Einstellung: Nur die letzten 15 Minuten
71          % Ermittlung des größten Amplitudenwertes aller Kanäle des
72          % Master-Moduls
73          tmpMax                          = ...
74              max(max(get(userData.hLine(1,1),'YData')));
75          % Einlesen des bisherigen Signalverlaufs
76          tmp                             = ...
77              get(userData.hPastSigPlot,'YData');
78          % Setzen des neuen Wertes für den bisherigen Signalverlauf
79          tmpShifted                      = ...
80              [tmp(2:end),tmpMax];
81          % Aktualisierung des bisherigen Signalverlauf
82          set(userData.hPastSigPlot, 'YData',tmpShifted);
83      end
84
85      % legendString        =  get(userData.master.Channel,'ChannelName');
86      % legend(userData.hAxes(1),legendString);
87      % set(userDatahLine(:,1),...
88      %     'Visible','On');
89      % set(hAxes(1),...
90      %     'Visible','On');
91
92      drawnow;
93      %% Wegschreiben der Ergebnisse
94      set(gcobject,'UserData',userData);
95  end
```

```
1  function plotData2(obj,~,gcobject)
2  %-----------------------------------------------------------------------
3  % Schnittstellenbeschreibung
4  %
5  % Autor:                Rick Plescher, 30.04.2015
6  %
7  % Funktionsaufruf:      plotData2(obj,~, gcobject)
8  %
```

```
9  % Fkt.-beschreibung:    Diese Funktion wird immer ausgeführt wenn der
10 %                       SamplesAcquiredFcnCount des Slave1-Modul erreicht
11 %                       wird.
12 %
13 % Eingabe:
14 %  aObject              : Objekt-Handle zu dem Slave1-Module mit seinen
15 %                         aktuellen Daten und EInstellungen
16 %
17 % Ausgabe:               keine direkte Ausgabe
18 %
19 %
20 % sontiges:
21 %                       Zum korrekten Ablauf ist die Function calcSPl.m
22 %                       notwendig. Außerdem muss das Fenster des bisherigen
23 %                       Signalverlaufs geöffnet sein.
24 %-------------------------------------------------------------------------
25 %% Programmcode
26 % Einlesen der aktuellen Daten
27 userData = get(gcobject,'UserData');
28 % Einlesen der aktuellen Mikrofondaten
29 readMicInput.file       = fullfile(fullfile(cd,'microphoneData.mat'));
30 readMicInput.header     = false;
31 userData.microphoneData = readMicrophoneData(readMicInput);
32 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
33 % Abfrage ob das Master-Module immer noch aktiv ist
34 if isrunning(obj)
35     data                = ... % Vorausschau der aktuellen erfassten Daten
36         peekdata(userData.slave1, userData.slave1.SamplesAcquiredFcnCount);
37     numActivechannels   = ... % Ermittlung der aktuell angemeldeten Kanäle
38         size(get(get(userData.slave1,'Channel'),'HwChannel'),1);
39
40     % Schleife zur Berechnung des pegelrichtige Zeitsignals
41     for n = 1:numActivechannels % Schleife maximal 1 bis 4
42         calcSplInput.signal      = data(:,n);
43         calcSplInput.sensitivity= userData.microphoneData{n+4,6};
44         set(userData.hLine(n,2) , 'YData',calcSpl(calcSplInput));
45     end
46 end
47 drawnow;
48
49 end
```

```
1  function plotData3(obj,~,gcobject)
2  %-------------------------------------------------------------------------
3  % Schnittstellenbeschreibung
4  %
5  % Autor:                 Rick Plescher, 30.04.2015
6  %
7  % Funktionsaufruf:       plotData3(obj,~, gcobject)
8  %
9  % Fkt.-beschreibung:     Diese Funktion wird immer ausgeführt wenn der
10 %                        SamplesAcquiredFcnCount des Slave3-Modul erreicht
11 %                        wird.
12 %
13 % Eingabe:
14 %  aObject               : Objekt-Handle zu dem Slave3-Module mit seinen
15 %                         aktuellen Daten und EInstellungen
16 %
17 % Ausgabe:               keine direkte Ausgabe
```

```
18 %
19 %
20 % sontiges:
21 %                        Zum korrekten Ablauf ist die Function calcSPl.m
22 %                        notwendig. Außerdem muss das Fenster des bisherigen
23 %                        Signalverlaufs geöffnet sein.
24 %-------------------------------------------------------------------------
25 %% Programmcode
26 % Einlesen der aktuellen Daten
27 userData = get(gcobject,'UserData');
28 % Einlesen der aktuellen Mikrofondaten
29 readMicInput.file       = fullfile(fullfile(cd,'microphoneData.mat'));
30 readMicInput.header     = false;
31 userData.microphoneData = readMicrophoneData(readMicInput);
32 %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
33 % Abfrage ob das Master-Module immer noch aktiv ist
34 if isrunning(obj)
35     data                = ...% Vorrausschau der aktuellen erfassten Daten
36         peekdata(userData.slave2, userData.slave2.SamplesAcquiredFcnCount);
37     numActivechannels   = ... % Ermittlung der aktuell angemeldeten Kanäle
38         size(get(get(userData.slave2,'Channel'),'HwChannel'),1);
39
40     % Schleife zur Berechnung des pegelrichtige Zeitsignals
41     for n = 1:numActivechannels % Schleife maximal 1 bis 4
42         calcSplInput.signal     = data(:,n);
43         calcSplInput.sensitivity= userData.microphoneData{n+8,6};
44         set(userData.hLine(n,3) , 'YData',calcSpl(calcSplInput));
45     end
46 end
47 drawnow;
48 end
```

```
1 function plotData4(obj,~,gcobject)
2 %-------------------------------------------------------------------------
3 % Schnittstellenbeschreibung
4 %
5 % Autor:                 Rick Plescher, 30.04.2015
6 %
7 % Funktionsaufruf:       plotData4(obj,~, gcobject)
8 %
9 % Fkt.-beschreibung:     Diese Funktion wird immer ausgeführt wenn der
10 %                        SamplesAcquiredFcnCount des Slave4-Modul erreicht
11 %                        wird.
12 %
13 % Eingabe:
14 %  aObject              : Objekt-Handle zu dem Slave4-Module mit seinen
15 %                          aktuellen Daten und EInstellungen
16 %
17 % Ausgabe:               keine direkte Ausgabe
18 %
19 %
20 % sontiges:
21 %                        Zum korrekten Ablauf ist die Function calcSPl.m
22 %                        notwendig. Außerdem muss das Fenster des bisherigen
23 %                        Signalverlaufs geöffnet sein.
24 %-------------------------------------------------------------------------
25 %% Programmcode
26 % Einlesen der aktuellen Daten
27 userData = get(gcobject,'UserData');
```

```
28 % Einlesen der aktuellen Mikrofondaten
29 readMicInput.file       = fullfile(fullfile(cd,'microphoneData.mat'));
30 readMicInput.header     = false;
31 userData.microphoneData = readMicrophoneData(readMicInput);
32 %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
33 % Abfrage ob das Master-Module immer noch aktiv ist
34 if isrunning(obj)
35     data                = ...% Vorrausschau der aktuellen erfassten Daten
36         peekdata(userData.slave3, userData.slave3.SamplesAcquiredFcnCount);
37     numActivechannels   = ... % Ermittlung der aktuell angemeldeten Kanäle
38         size(get(get(userData.slave3,'Channel'),'HwChannel'),1);
39
40     % Schleife zur Berechnung des pegelrichtige Zeitsignals
41     for n = 1:numActivechannels % Schleife maximal 1 bis 4
42         calcSplInput.signal      = data(:,n);
43         calcSplInput.sensitivity= userData.microphoneData{n+12,6};
44         set(userData.hLine(n,4) , 'YData',calcSpl(calcSplInput));
45     end
46 end
47 drawnow;
48 end
```

```
1 function microphoneData = readMicrophoneData(aInput)
2 %-------------------------------------------------------------------------
3 % Schnittstellenbeschreibung
4 %
5 % Autor:                Rick Plescher, 30.04.2015
6 %
7 % Funktionsaufruf:      microphoneData = readMicrophoneData(aInput)
8 %
9 % Fkt.-beschreibung:    Die Funktion liest die Daten der aktuell
10 %                      eingestellten Mikrofone ein. Es ist möglich die
11 %                      Daten mit oder ohne Header einzulesen.
12 %
13 % Eingabe:
14 %  aInput.
15 %   .file              : String mit dem Speicherort der
16 %                        Mikrofondaten-Datei
17 %   .header            : Stellt ein Flag zu Abfrage des Headers dar. Kann
18 %                        die Werte 'true' oder 'false' haben
19 %
20 % Ausgabe:
21 %   microphoneData     : Cell-Array mit den Daten über alle aktuell
22 %                        hinterlegten Mikrofone
23 %
24 % sontiges:
25 %
26 %-------------------------------------------------------------------------
27 %% Programmcode
28 if aInput.header
29     tmp = load(aInput.file); % Laden des aktuellen Mikrofon-Files
30     microphoneData = tmp.microphoneData; % Einlesen der Mikrofondaten
31                                          % inklusive Header
32 else
33     tmp = load(aInput.file); % Laden des akutellen Mikrofon-Files
34     microphoneData = tmp.microphoneData(2:end,:); % Einlesen der Daten ohne
35                                                   % Header
36 end
37 %-------------------------------------------------------------------------
```

```
function refreshMicData_callback(aTableHandle,aTableInfo)
%-------------------------------------------------------------------------
% Schnittstellenbeschreibung
%
% Autor:                Rick Plescher, 30.04.2015
%
% Funktionsaufruf:      refreshMicData_callback(aTableHandle,aTableInfo)
%
% Fkt.-beschreibung:    Die Funktion ist ein Callback der
%                       editMicrophoneData.m Funktion, wenn die
%                       Seriennummer in der Tabelle geändert wurden.
%
% Eingabe:
%  aTableInfo           : Struktur mit Informationen über die geänderte
%                         Tabellen-Zelle
%   .EditData           : Wert der editierten Tabellen-Zelle
%   .Indices            : Vektor mit der relativen Position des editierten
%                         Tabellen-Elements innerhalb der Tabelle
%
% Ausgabe:              keine direkte Ausgabe.
%                       Nach Aufruf der Funktion wird die
%                       microphoneData.mat im aktuellen Pfad mit den neuen
%                       Daten aktualisiert
%
% sontiges:             Der Aufruf kann nur als Callback erfolgen und es
%                       muss eine microphoneData.mat Datei im aktuellen
%                       Pfad liegen.
%
%-------------------------------------------------------------------------
%% Einlesen der Mikrofondaten der gesamten Datenbank
readMicInput.file   = ...
   fullfile(cd,'microphoneDatabase','completeMicDatabase-DO NOT EDIT.mat');
readMicInput.header = true;

microphoneData      = readMicrophoneData(readMicInput);
% Die Daten der Struktur microphoneData enthalten hier den Header mit den
% Items. Daher liegen die Daten der Mikrofone in den Zeilen 2:end
%-------------------------------------------------------------------------
%% Programmcode
% Schleife über die Gesamt-Mikrofondatenbank
for n =2:size(microphoneData,1)
    % Abfrage ob das editierte Element gleich einem Element in der
    % Datenbank ist.
    if(strcmp(microphoneData{n,5},aTableInfo.EditData))
        % Laden der editierten Tabelle
        oldTableData                            = get(aTableHandle,'Data');
        % Initialisierung der neuen Tabellen-Matrix
        newTableData                            = oldTableData;
        % Einfügen der neuen editierten Daten in der Tabellen-Matrix+++++++
        % Aktualisierung der Modell-Bezeichnung
        newTableData(aTableInfo.Indices(1),1)   = microphoneData(n,4);
        % Aktualisierung des Mikrofon-Übertragungsfaktor
        newTableData(aTableInfo.Indices(1),3)   = microphoneData(n,6);
        % Aktualisierung des Datums des Mikrofon-Übertragungsfaktor
        newTableData(aTableInfo.Indices(1),4)   = microphoneData(n,7);
        %++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        % Aktualisierung der Tabellen-Daten
        set(aTableHandle,'Data',newTableData);
    elseif(strcmp(aTableInfo.EditData,' ')) % kein Mikrofon vorhanden
        % Laden der editierten Tabelle
```

```
61          oldTableData                                = get(aTableHandle,'Data');
62          % Initialisierung der neuen Tabellen-Matrix
63          newTableData                                = oldTableData;
64          % Einfügen der neuen editierten Daten in der Tabellen-Matrix+++++++
65          % Aktualisierung der Modell-Bezeichnung
66          newTableData(aTableInfo.Indices(1),1)    = {' '};
67          % Aktualisierung des Mikrofon-Übertragungsfaktor
68          newTableData(aTableInfo.Indices(1),3)    = {NaN};
69          % Aktualisierung des Datums des Mikrofon-Übertragungsfaktor
70          newTableData(aTableInfo.Indices(1),4)    = {' '};
71          %+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
72          % Aktualisierung der Tabellen-Daten
73          set(aTableHandle,'Data',newTableData);
74      end
75  end
76
77  % Neue Initialisierung der Tabelle mit den neuen Daten und dem Header
78  tmp = cell(size(newTableData,1),7);
79  for n = 1:size(newTableData,1)
80      tmp(n,:) = ...
81          {n                   ,... % Lfd. Nr.
82           n                   ,... % Kanalzahl
83           mod(n-1,4)          ,... % Hardware-Kanalzahl
84           newTableData{n,1}   ,... % Modell-Bezeichnung
85           newTableData{n,2}   ,... % Seriennummer
86           newTableData{n,3}   ,... % Mikrofon-Übertragungsfaktor
87           newTableData{n,4}};      % Datum der Kalibrierung
88  end
89
90  % Header
91  microphoneItems = { ...
92          'Num'            ,'Ch-Num'        ,'HwCh-Num' ,'Model',...
93          'Serialnumber'   ,'Sensitivity'   ,'Last Calibration'};
94  microphoneData  = [microphoneItems;tmp];
95  %-------------------------------------------------------------------------
96  %% Wegschreiben der Ergebnisse
97  save(fullfile(cd,filesep,'microphoneData.mat'),'microphoneData');
98  %-------------------------------------------------------------------------
99  end
```

```
1   function saveDAQMicrophoneData(aInput)
2   %-------------------------------------------------------------------------
3   % Schnittstellenbeschreibung
4   %
5   % Autor:                Rick Plescher, 30.04.2015
6   %
7   % Funktionsaufruf:      saveDAQMicrophoneData(aInput)
8   %
9   % Fkt.-beschreibung:    Soeichert die aktuelle Mikrofoneinstellung als mat
10  %                       im cd Ordner
11  %
12  % Eingabe:
13  %   aInput:
14  %       .header:        Mit oder ohne Grßenbeschreibung
15  %       .file:          Speicherpfad
16  %
17  %
18  % Ausgabe:
19  %
```

```
20 %
21 % sontiges:
22 %
23 %-------------------------------------------------------------------------
24 %% Programmcode
25 if aInput.header
26     readMicInput.file       = fullfile(cd,'microphoneData.mat');
27     readMicInput.header     = aInput.header;
28     microphoneData          = readMicrophoneData(readMicInput);
29
30     save(fullfile(aInput.saveFileStr)       ,'microphoneData');
31     save(fullfile(cd,'microphoneData.mat')  ,'microphoneData');
32 else
33     readMicInput.file       = fullfile(cd,'microphoneData.mat');
34     readMicInput.header     = aInput.header;
35     microphoneData          = readMicrophoneData(readMicInput);
36
37     save(fullfile(aInput.saveFileStr)       ,'microphoneData');
38     save(fullfile(cd,'microphoneData.mat')  ,'microphoneData');
39 end
40 %-------------------------------------------------------------------------
41 %% Wegschreiben der Ergebnisse
42
43 %-------------------------------------------------------------------------
```

```
1  function savePresetMicData_mouseCallback(~,~,aTableHandle)
2  %-------------------------------------------------------------------------
3  % Schnittstellenbeschreibung
4  %
5  % Autor:                Rick Plescher, 30.04.2015
6  %
7  % Funktionsaufruf:      savePresetMicData_mouseCallback(~,~,aTableHandle)
8  %
9  % Fkt.-beschreibung:    Speichert die in der aktuelle Mikrofontabelle
10 %                       hinterlegt sind.
11 %
12 % Eingabe:
13 % aTabelHandle:         Handle zu Tabelle der editMicrphoneData()-funktion
14 %
15 % Ausgabe:              keine direkte
16 %                       indirekt wird eine neues Setup gespeichert.
17 %
18 %
19 % sontiges:
20 %
21 %-------------------------------------------------------------------------
22 %% Programmcode
23  [fileName,pathName] = uiputfile(...
24         '.mat',...
25         'Please save your new microphone setup file');
26
27     microphoneItems = { ...
28         'Num'           ,'Ch-Num'       ,'HwCh-Num' ,'Model',...
29         'Serialnumber'  ,'Sensitivity'  ,'Last Calibration'};
30     if fileName~=0
31     data            = get(aTableHandle,'Data');
32     tmp             = cell(size(data,1),7);
33     for n = 1:size(data,1)
34         tmp(n,:)            = ...
```

```
35                {n,n,mod(n-1,4),data{n,1},data{n,2},data{n,3},data{n,4}};
36        end
37        microphoneData  = [microphoneItems;tmp];
38
39        save([pathName,fileName],'microphoneData');
40
41        delete(aTableHandle);
42        else
43            warndlg('No new Mic-Preset saved!');
44        end
45    %-----------------------------------------------------------------------
46    %% Wegschreiben der Ergebnisse
47    %-----------------------------------------------------------------------
48    end
```