Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

„Einflussfaktoren auf die Lokalisationsgüte von Passive Audio Monitoring Systems (PAMS)“

Unter Betrachtung der Lokalisation als Optimierungsproblem

vorgelegt von

B.Sc. Florian Hoedt

104219, UNIGIS MSc Jahrgang 2015

Zur Erlangung des Grades
„Master of Science (Geographical Information Science & Systems) – MSc(GIS)”

Salzburg, 01.04.2018

Abstract

This thesis investigated the correlation between Passive Audio Monitoring Systems (PAMS) algorithms for classification and localization and sound obstacles, the direction of the sound and in vitro songs of three different species. Four hypotheses were analyzed:

1. The sound classification accuracy positively correlates with the localization accuracy
2. The localization accuracy is determined by the number of obstacles between signal source and microphone (sound shading)
3. The localization accuracy is determined by the type and kind of sound (species, song type)
4. The localization accuracy is determined by the direction of the sound

An of four Raspberry Pi consisting PAMS was installed and a speaker was used to play songs of textitAlauda arvensis, *Carduelis carduelis* und *Phylloscopus collybita* at five different signal locations. The setup and signal locations were surveyed with a tachymeter. The used recording logic was programmed in *Python* by using the *pyalsaaudio* package. The Network Time Protocol was used to synchronize the DateTime stamped WAV files. The spectrogram cross-correlation classification was performed with the *R* package *monitoR*. To localize the signals a *SciPy.minimize* based optimization algorithm was used. To measure the degree of sound shading LIDAR data from the DSM North-Rhine Westfalia was used.

The five signal sources yielded a total of 15.412 classified and localized data points. The classification resulted in ∼14.8 % *Phylloscopus collybita* and ∼15.5 % *Alauda arvensis* false-positives, and 100 % false-negatives for *Carduelis carduelis*. The

Carduelis carduelis samples have been wrongly classified as *Phylloscopus collybita* (∼81.5 %) and *Alauda arvensis* (∼18.5 %). Localization accuracy was meased as $\bar{x}\ 34.96\ \pm 19.49\,m$ (min: 1.1, max: 95.8 m) with clusters at 10 and 45 m at different signal locations and sound directions.

A very weak negative correlation between classification and localization accuracy was determined (Pearson's product-moment correlation, ∼-0.016, p-value 0.042). No significant correlation could be determined between sound shading and localization accuracy (Pearson's product-moment correlation -0.29, p-value $< -2.2^{-16}$). The species had a significant impact on location accuracy (Kruskal-Wallis rank sum test, p-Value 1.6^{-7}). The direction of the sound had no significant impact on location accuracy (Pearson's product-moment correlation -0.07, p-value $< -2.2^{-16}$).

Zusammenfassung

Die vorliegende Thesis untersucht die Zusammenhänge zwischen Passive Audio Monitoring System (PAMS) Algorithmen zur Klassifikation sowie Lokalisation und schallverschattenden Objekten, der Schallausrichtung und den genutzten in vitro Vogellauten drei verschiedener Vogelarten. Im Detail werden vier Hypothesen geprüft:

1. Die Güte der Klassifikation korreliert positiv mit der Güte der Lokalisation.
2. Die Güte der Lokalisation ist abhängig von:
   - 2.a Der Schallverschattung des Schallgebers zum Mikrofon,
   - 2.b Der Art des Schalls,
   - 2.c Der Richtung des Schalls.

Zur Untersuchung wurden ein aus vier *Raspberry Pis* bestehendes PAMS installiert und eingemessen. An Fünf ebenfalls eingemessenen Lautsprecherpositionen wurden jeweils Strophen der Vogelarten *Alauda arvensis*, *Carduelis carduelis* und *Phylloscopus collybita* abgespielt. Die Aufnahmelogik wurde in *Python* unter Verwendung des *pyalsaaudio* Packages programmiert und die Aufnahmen als WAV-Dateien mit über NTP synchronisiertem Zeitstempel gespeichert. Für Klassifikation der aufgenommenen Daten wurde die spectrogramm cross correlation des *R* package *monitoR* genutzt. Die Lageverortung wurde als Optimierungsproblem behandelt und über das *SciPy.minimize* Modul durchgeführt. Als Maß der Schallverschattung von Signalposition zu den Sensoren wurden die Laserscan Returns des Digitalen Oberflächenmodells NRW verwendet.

Insgesamt konnten für die fünf Signalpositionen 15.412 Vogellaute erfasst werden. Hierbei sind ∼14.8 % der *Phylloscopus collybita*, ∼15.5 % der textitAlauda arvensis

sowie 100 % der *Carduelis carduelis* Detektionen falsch klassifiziert worden. Die *Carduelis carduelis* Detektionen wurden zu ∼81.5 % als *Phylloscopus collybita* und zu ∼18.5% als *Alauda arvensis* klassifiziert. Die Lokalisation ergab Abweichungen vom Signal zum verortetem Punkt von $\bar{x}\ 34.96\ \pm 19.49\,m$ (min: 1.1, max: 95.8 m), mit Häufungen bei 10 und 45 m Abweichung zu unterschiedlicher Zeit und an verschiedenen Abspielpositionen.

Es konnte eine sehr schwache negative Korellation zwischen der Güte der Klassifikation und der Güte der Lokalisation festgestellt werden (Pearson's product-moment correlation ∼-0.016, p-value 0.042). Zwischen der Schallverschattung und der Verortungsgenauigkeit konnte eine signifikante negative Korellation ermittelt werden (Pearson's product-moment correlation -0.29, p-value $< -2.2^{-16}$). Die abgespielte Vogelart hat einen signifikanten Einfluß auf die Lageverortungsgüte (Kruskal-Wallis rank sum test, p-Value 1.6^{-7}). Für die Abspielrichtung der Vogellaute konnte keine maßgebliche Korellation zur Güte der Lokalisierung festgestellt werden (Pearson's product-moment correlation -0.07, p-value $< -2.2^{-16}$).

Eigenständigkeitserklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen worden ist.
Die Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Höxter, den 01. 04. 2018 Florian Hoedt

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Akronyme

CBA Component-Based-Architecture. 53

DOA Direction of Arrival. 8, 9

FDOA Frequency Difference of Arrival. 7

GMM Gaussian Mixture Models. 6, 7

HMM Hidden Markov Models. 6, 7

HRTF Head Related Transfer Function. 7

HS OWL Hochschule Ostwestfalen-Lippe. 12

MFCC Mel Frequency Cepstral Coefficients. 7

NTP Network Time Protocoll. 16, 47

ORM Object-Relational-Mapping. 53

OS Operation System. 16, 45

PAMS Passive Audio Monitoring Systems. 1–3, 5, 12, 14, 25, 45, 53

SVM Support Vector Machine. 6, 7

TDOA Time Difference of Arrival. 7, 8, 27, 50

1 | Einleitung

1.1 Ausgangssituation

Das Monitoring der Umweltsituation ist in vielen Ländern gesetzlich vorgeschrieben, so zum Beispiel in ausgewiesenen Schutzgebieten (BUNDESREPUBLIK DEUTSCHLAND, 2009; EUROPEAN COMMISSION - GENERAL DIRECTION ENVIRONMENT, 2007). In einigen Sektoren der Umweltbeobachtung sind Sensoren bereits seit vielen Jahren die Standardmethode, um Monitoring durchzuführen, so beispielsweise zur Aufnahme chemischer Parameter der Luft und des Wassers. Motiviert vom Monitoringbedarf und ermöglicht durch den technischen Fortschritt der letzten Jahre wird im Bereich der low-cost Hardware zunehmend geforscht inwiefern bisher manuell durchgeführtes Monitoring mit Sensoren abgebildet werden kann (SHONFIELD & BAYNE, 2017; HILL ET AL., 2017; BROWNING ET AL., 2017).

Entsprechend ist, wie in SHONFIELD & BAYNE (2017) und Abbildung 1.1 dargestellt, in den letzten Jahren das wissenschaftliche Interesse an sogenannten Passive Audio Monitoring Systems (PAMS) zum Umweltmonitoring gestiegen. Darüber hinaus wird in AIDE ET AL. (2013) für die Notwendigkeit solcher Systeme argumentiert. Als Hauptargumente werden dort erhöhter Monitoringbedarf durch sich rasch ändernden Ökosysteme, die Wiederholbarkeit der Analysen durch aufgezeichnete Soundsamples und die Eliminierung von Kartierer-Bias angegeben werden. MENNILL ET AL. (2012); SHONFIELD & BAYNE (2017) ergänzen hierbei um folgende Aspekte:

- Die Möglichkeit räumlichen Kontext über Lokalisierungsalgorithmik herzustellen,

- Im Vergleich zu traditionellen Methoden geringe Invasivität,
- Mehreren Individuen können simultan aufgenommen werden,
- Aufnahmen sind über große Flächen und lange Zeiträume möglich,
- Erfassungen sind unabhängiger von den Sichtverhältnissen (Nacht, dichte Vegetation).

Zudem sieht die europäische Kommission vor, mittel- bis langfristig eine Harmonisierung der Aufnahmemethodik des Monitorings anzustreben (EUROPEAN COMMISSION - GENERAL DIRECTION ENVIRONMENT, 2007, S. 18). Der Ressortforschungsplan 2018 des BMUB (2017, S. 44) wünscht die „Fachliche Weiterentwicklung von Instrumenten insbesondere in den Bereichen Monitoring [...]“.

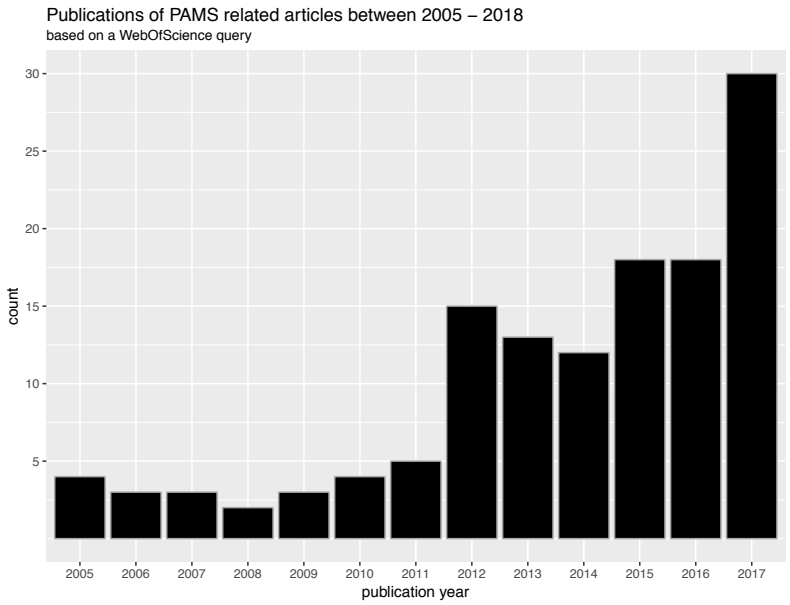

Abbildung 1.1: Zwischen 2005 und 2017 veröffentlichte Artikel über Audio Monitoring Systeme. Die verwendete Abfrage ist unter Abschnitt A.1 auf Seite 59 gezeigt.

1.2 Motivation

Um das volle Potential von PAMS auszuschöpfen und die Vorteile der Systeme nutzen zu können besteht weiterer Forschungsbedarf: So schränkt SHONFIELD & BAYNE (2017) ein, dass bisher Vogeldichten oftmals unzureichend zu ermitteln sind. Der tatsächliche Detektionsradius der Systeme ist durch Unterschiede in

der Hardware, Software und der Lautstärke von Vogelart zu Vogelart nur schwer zu erfassen. Somit ist unklar auf welches Gebiet sich die jeweiligen Detektionen beziehen. Für eine Auswertung müssen entsprechend weitere Methoden verwendet werden, wie zum Beispiel die Identifikation von Lautstärkepegeln in Relation zur Distanz durch in vitro Wiedergabe von Vogelarten (LAMBERT & MCDONALD, 2014). Eine andere in SHONFIELD & BAYNE (2017) als vielversprechend genannte Methode, um dieses Problem und weitere ökologische Fragestellungen zu lösen, ist die Lokalisation der jeweilig detektierten Vogellaute.

Zur Beantwortung landschaftsökologischer Fragestellungen sollten PAMS in die freie Landschaft installiert werden. Dabei stellt sich die Frage, inwiefern die Schallverschattung durch Vegetation und anderer Objekte, die Güte der Klassifikation und Lokalisation beeinflusst. Bisherige Untersuchungen von MENNILL ET AL. (2006) zeigen keine signifikanten Auswirkungen der Vegetation auf die Lokalisations- und Klassifikationsgenauigkeit unter tropischem Regenwald. Des Weiteren konnten LAMBERT & MCDONALD (2014) keine signifikanten Unterschiede in der Abnahme der Lautstärke in Relation zur Distanz zwischen den untersuchten Vegetationstypen ermitteln. Der dort dominante Unterwuchs bestand jedoch aus Stauden (*Lantana camara*) und es wurde lediglich die Lautstärkeabnahme, nicht jedoch der Klassifizierungs- oder Lokalisierungserfolg, betrachtet.

Daher verbleibt die Frage, welchen Einfluß mitteleuropäische Vegetation verschiedener Ausprägung auf die Klassifikation und Lokalisation hat. Durch ausreichende Kenntnis über diese Effekte könnte in Zukunft die Platzierung von PAMS Komponenten optimiert und die verwendeten Algorithmen angepasst werden.

1.3 Aufgabenstellung

Die vorliegende Thesis erforscht die Zusammenhänge zwischen schallverschattenden Objekten, wie Vegetation, zu den genutzten PAMS Algorithmen zur Klassifikation bzw. Lokalisation von Vogellauten. Hierbei wird vermutet, dass schallverschattende Objekte die Güte der Algorithmen beeinträchtigen. Im Rahmen der Arbeit sollen daher folgende Thesen geprüft werden:

1. Die Güte der Klassifikation korreliert positiv mit der Güte der Lokalisation.

2. Die Güte der Lokalisation (und bei Verifizierung von These 1 auch Klassifikation) ist abhängig von:

   2.a Der Schallverschattung des Schallgebers (Vogel) zum Mikrofon,

   2.b Die Art des Schalls (Vogelart),

   2.c Die Richtung des Schalls (Kopfausrichtung des Vogels).

Die Untersuchung der Hyphothesen 2.b, sowie 2.c sind notwendig, um die angenommenen Effekte dieser Parameter bei der Untersuchung nach Schallverschattung ausschließen zu können. Die Prüfung dieser Hypothesen lässt zu, die Leitfrage hinreichend genau zu beantworten.

1.4 Struktur der Thesis

Das Kapitel Grundlagen (2) vermittelt einen Überblick über die derzeit im Themengebiet dieser Arbeit verwendeten Methoden und Hardwarekomponenten. Die für die Beantwortung der Aufgabenstellung verwendeten Methoden sind unter Kapitel 3 näher beschrieben und deren Auswahl argumentiert. Im Kapitel 4 werden die erfassten Daten und die Ausgaben der verwendeten Algorithmen in Tabellen und Diagrammen dargestellt und bezüglich der aufgestellten Forschungsfragen statistisch ausgewertet. Das Kapitel Diskussion (5) hinterfragt die verwendeten Methoden und Ergebnisse kritisch. Abschließend stellt der Ausblick (6) Anpassungmöglichkeiten der verwendeten Methodik dar und der weitere Forschungsbedarf wird aufgezeigt. Der verwendete Programmcode und die erstellten Datensätze sind im Anhang gelistet.

2 | Grundlagen

Das Kapitel Grundlagen vermittelt einen Überblick über die derzeit im Themengebiet dieser Arbeit verwendeten Methoden und Hardwarekomponenten. Die für die Beantwortung der Aufgabenstellung (Abschnitt 1.3 auf Seite 3) verwendeten Methoden sind unter Kapitel 3 auf Seite 12 näher beschrieben.

2.1 Passive Audio Monitoring Systeme (PAMS)

PAMS bestehen aus Sensoren für Audioaufnahmen, mit unterschiedlich ausgeprägtem Maß an eigener Logik. Die für diese Systeme verwendete Hardware ist divers und wird z.B. über *Raspberry Pis* (RASPBERRY.ORG, 2018), *Song Meter* (WILDLIFE ACOUSTICS, 2018) oder *Mac Minis* realisiert (siehe (AIDE ET AL., 2013; DIGBY ET AL., 2013; BROWNING ET AL., 2017; WHYTOCK & CHRISTIE, 2017)). Neben proprietären (HEDLEY ET AL., 2017) oder closed-source (AIDE ET AL., 2013) Systemen, sind in letzter Zeit immer häufiger open-source und low-cost Systeme beschrieben und getestet worden (WHYTOCK & CHRISTIE, 2017; HILL ET AL., 2017).

2.2 Klassifikation von Audiodaten

Die von PAMS verwendeten Klassifikatoren versuchen über verschiedenartige Matching Algorithmen vorher erstellte Regeln in einer Audiodatei zu detektieren (SHONFIELD & BAYNE, 2017). Die Anzahl der verfügbaren Matching Algorithmen steigt und ist vielfältig, so zum Beispiel:

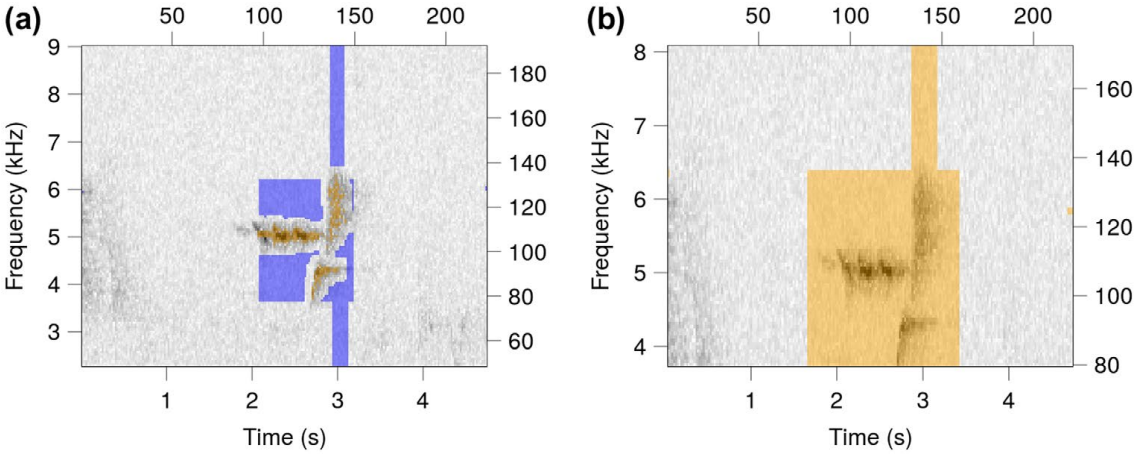

Abbildung 2.1: Beispiele für ein binary point template (a) und cross correlation template (b) — aus KATZ ET AL. (2016b, S. 202).

- Band Limited Energy Detector (MILLS, 2000)
- Binary Point Matching (KATZ ET AL., 2016b)
- Spectrogramm Cross Correlation (KATZ ET AL., 2016b)
- Decision Trees (DIGBY ET AL., 2013; ACEVEDO ET AL., 2009; ROSS & ALLEN, 2014)
- Support Vector Machine (SVM) (ACEVEDO ET AL., 2009)
- Hidden Markov Models (HMM) / Gaussian Mixture Models (GMM) (RANJARD ET AL., 2017; VENTURA ET AL., 2015; AIDE ET AL., 2013; KOGAN & MARGOLIASH, 1998)

Hierbei lassen sich die Klassifikatoren grundsätzlich in zwei Typen, template- oder feature-basiert, aufteilen (ZHAO ET AL., 2017).

Erstere vergleichen das vorher manuell erstellte template mit dem erfassten Spektrogramm, z.B. über binary point matching oder spectrogramm cross correlation (KATZ ET AL., 2016b). Für binary point matching templates werden Bereiche als on/off Punkte erfasst, wie in Abbildung 2.1 (a) dargestellt - orange Punkte zeigen on Points, blaue Punkte off Points. Hierbei wird zur Klassifizierung angenommen, dass in den orangen Zeit/Frequenzbereichen hohe Amplituden vorherrschen und in den blauen niedrige bis keine. Für das cross correlation matching werden ganze Bereiche (siehe Abbildung 2.1 (b)) des Spektrogramms verglichen. Das Matching

wird über eine über die Zeitachse gleitenden Abgleich zwischen template und zu klassifizierender Daten durchgeführt. Hierbei gibt der Korellationskoeffizient die „Gleichheit“ zwischen template und zu analysierenden Teil des Datensatzes an und reicht von -1.0 bis 1.0 (KATZ ET AL., 2016b), wie beispielhaft in Abbildung 3.9 auf Seite 20 dargestellt.

Die feature-basierten Klassifikatoren berechnen verschiedene spektro-temporale Audiomerkmale der Vorlage. Einige Beispiele dieser Merkmale sind der spektrale Centroid, die Bandbreite des Signals, oder die Mel Frequency Cepstral Coefficients (MFCC) (ZHAO ET AL., 2017; GIANNAKOPOULOS, 2015). Diese Wertpaare werden nachfolgend mittels Cluster Algorithmen (Decision Trees, SVM, HMM, GMM) zu den jeweiligen Spezies zugeordnet, oder verworfen.

Darüberhinaus werden über Forschungswettbewerbe, wie den LifeCLEF Bird Identification Task, bestehende Algorithmen angepasst und neue erstellt sowie evaluiert (GOËAU ET AL., 2016; LASSECK, 2015). Während die Auswahl an Klassifikatoren groß ist, sind jedoch nur einige dieser Algorithmen anwendungsbereit und unter open-source Lizenz veröffentlicht (SHONFIELD & BAYNE, 2017).

2.3 Lokalisierung von Soundquellen

Um erkannte Ereignisse zu Lokalisieren können verschiedene Algorithmen verwendet werden. Grundsätzlich werden Zeit-, Frequenz-, und/oder Amplitudenunterschiede als Messgrößen zur Verortung verwendet (IMRAN ET AL., 2016; LI ET AL., 2016; ALI ET AL., 2009). Einige Algorithmen liefern lediglich 2D, andere eine 3D Lokalisation, wie in LI ET AL. (2016) tabellarisch zusammengefasst.

Die Erfassung der Zeit-, Frequenz- und/oder Amplitudeuntschiede sind über verschiedene Mikrofonaufbauten möglich. HORNSTEIN ET AL. (2006) nutzt ein dem menschlichen Ohr nachempfundenes Setup, um mit vorher bekannten Frequency Difference of Arrival (FDOA) und Time Difference of Arrival (TDOA) eine Head Related Transfer Function (HRTF) zur Verortung zu nutzen.

BUAKA MUANKE & NIEZRECKI (2007) nutzen TDOA zur Lösung einer nichtlinearen, hyperbolischen Gleichung, um die Soundquelle zu verorten, wie in Abbildung 2.2 auf der nächsten Seite gezeigt. Der Schnittpunkt der jeweils berechneten Hyperbolen gibt hierbei die Position der Soundquelle an.

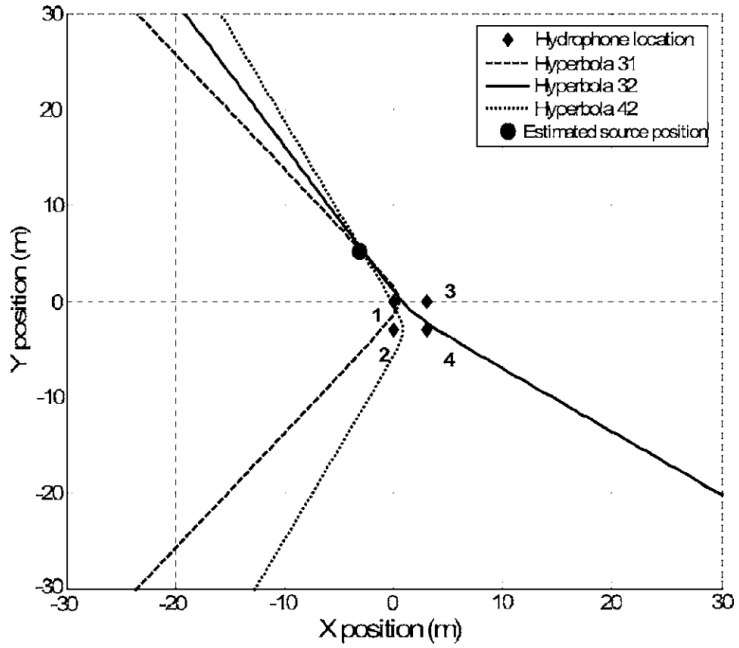

Abbildung 2.2: Errechnete sich schneidende Hyberbolen zur Verortung einer Soundquelle. Die Hyperbole 31 wurde zwischen dem Hydrophone 3 und 1, die Hyperbole 32 und 42 respektive, berechnet — aus BUAKA MUANKE & NIEZRECKI (2007, S. 6).

Der von ALI ET AL. (2009) genutzte Maximum-Likelihood-Algorithmus nutzt die über TDOA berechnete Direction of Arrival (DOA) zur Verortung, wie exemplarisch in Abbildung 2.3 auf der nächsten Seite dargestellt.

Der in dieser Arbeit getestete und verwendete Lokalisierungsalgorithmus betrachtet die Positionsfindung als Optimisierungsproblem, siehe Abbildung 2.4 auf Seite 10. Der Algorithmus basiert auf dem *SciPy.minimize* Modul (THE SCIPY COMMUNITY, 2017), wie in Abschnitt A.4 auf Seite 66 gelistet.

Gegeben seien die Mikrofonpositionen M_n, sowie die Signalposition S. Hierbei ist die Mikrofonposition M_A dem Signal S am nähesten. Für die Mikrofonpositionen $M_{n'} \rightarrow_{n''}$ können somit, wie in Gleichung (2.1) beschrieben, die TDOA berechnet werden.

$$TDOA_{M_n} = \overrightarrow{S\ M_n} - \overrightarrow{S\ M_A} \tag{2.1}$$

Für die gesuchte Signalposition SP gilt hierbei Gleichung (2.2) auf der nächsten Seite.

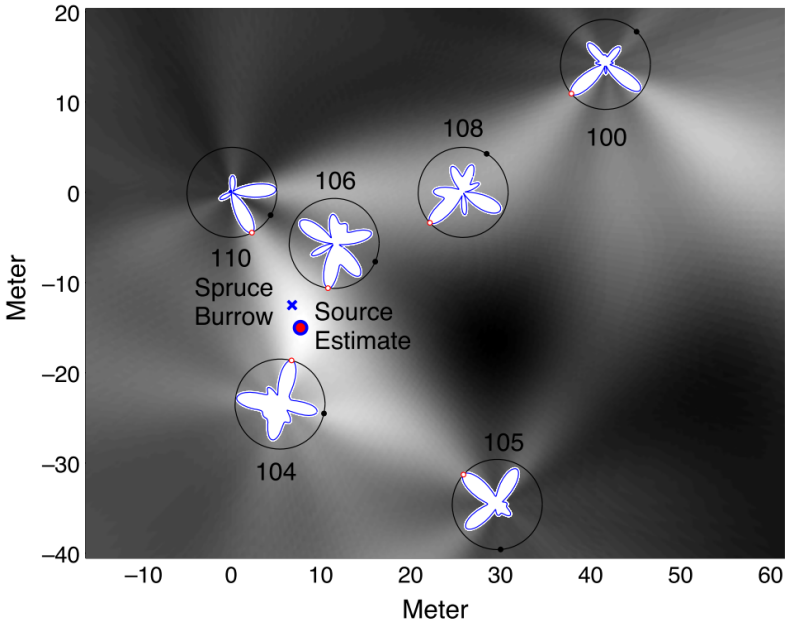

Abbildung 2.3: Errechnete pseudo-likelihood Karte der aus DOA errechneten Position (Source Estimate) der Schallquelle (Spruce Burrow). Die Umkreise stellen Sensorpositionen dar, die jeweiligen Schlaufen die errechneten DOA Schätzungen mit roter Markierung beim Winkelmaß der größten Wahrscheinlichkeit. Der Hintergrund stellt die summierte Wahrscheinlichkeit der extrapolierten DOA Schätzungen, von niedrig Schwarz bis hoch als Weiß, dar — aus ALI ET AL. (2009, S. 14).

$$S_{[x,y]} \mapsto SP_{[x,y]fit} = 0 \tag{2.2}$$

$SP_{[x,y]fit}$ wiederum entspricht der Standardabweichung der TDOA zu den jeweiligen Mikrofonpositionen, wie in Gleichung (2.3) gezeigt und beispielhaft in Abbildung 2.4 auf der nächsten Seite dargestellt.

$$SP_{[x,y]fit} = \sigma \left\| \begin{matrix} \overrightarrow{SP_{[x,y]}\ M_n} - TDOA_{M_n} \\ \vdots \end{matrix} \right\| \tag{2.3}$$

Die hier beschriebene Abhängigkeit der $SP_{[x,y]fit}$ basierten Fitnessfunktion wurde über eine Simulation getestet und ausgewertet. Der Zusammenhang konnte, wie in Abbildung 2.5 auf Seite 11 dargestellt, mit einer Pearson's product-moment correlation von 0.8 sowie einem p-value von $< 2.2^{-16}$ nachgewiesen werden.

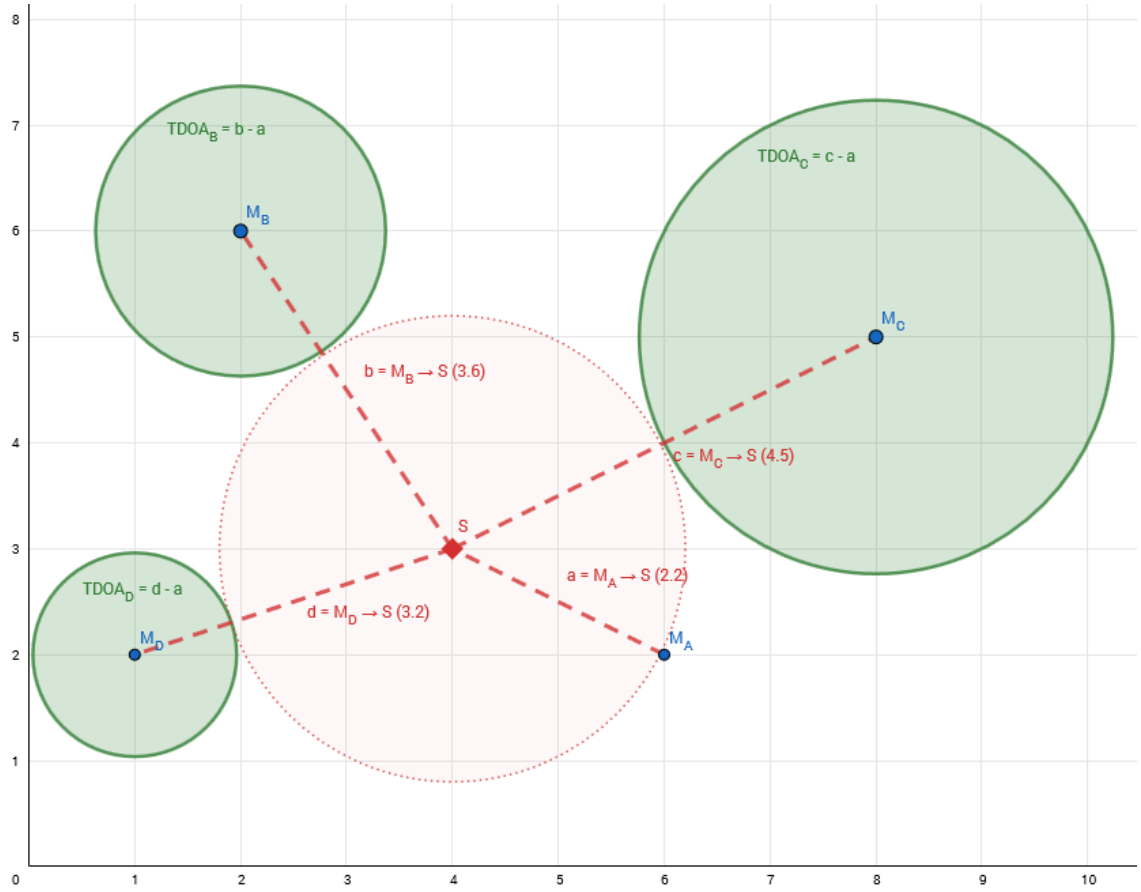

Abbildung 2.4: Beispielaufbau des auf Standardabweichung (σ) minimierenden Lokalisationsalgorithmus. Gegeben seien die vier Sensoren $M_A \dots M_D$, sowie die Signalposition S. Die grünen Umkreise enstsprechen den über Gleichung (2.1) auf Seite 8 berechneten TDOA. Durch über den in Abschnitt A.4 auf Seite 66 beschriebemen Lokalisierungsalgorithmus werden zufällig Punkte eingestreut und auf $SP_{[x,y]fit}$ minimiert (siehe Gleichung (2.3) auf der vorherigen Seite). Für den Punkt S gilt Gleichung (2.2) auf der vorherigen Seite, da $TDOA_n - \overrightarrow{SP_{[x,y]}\ M_n}$ für alle vier Sensoren 2.2 ergibt und σ ensptrechend 0.

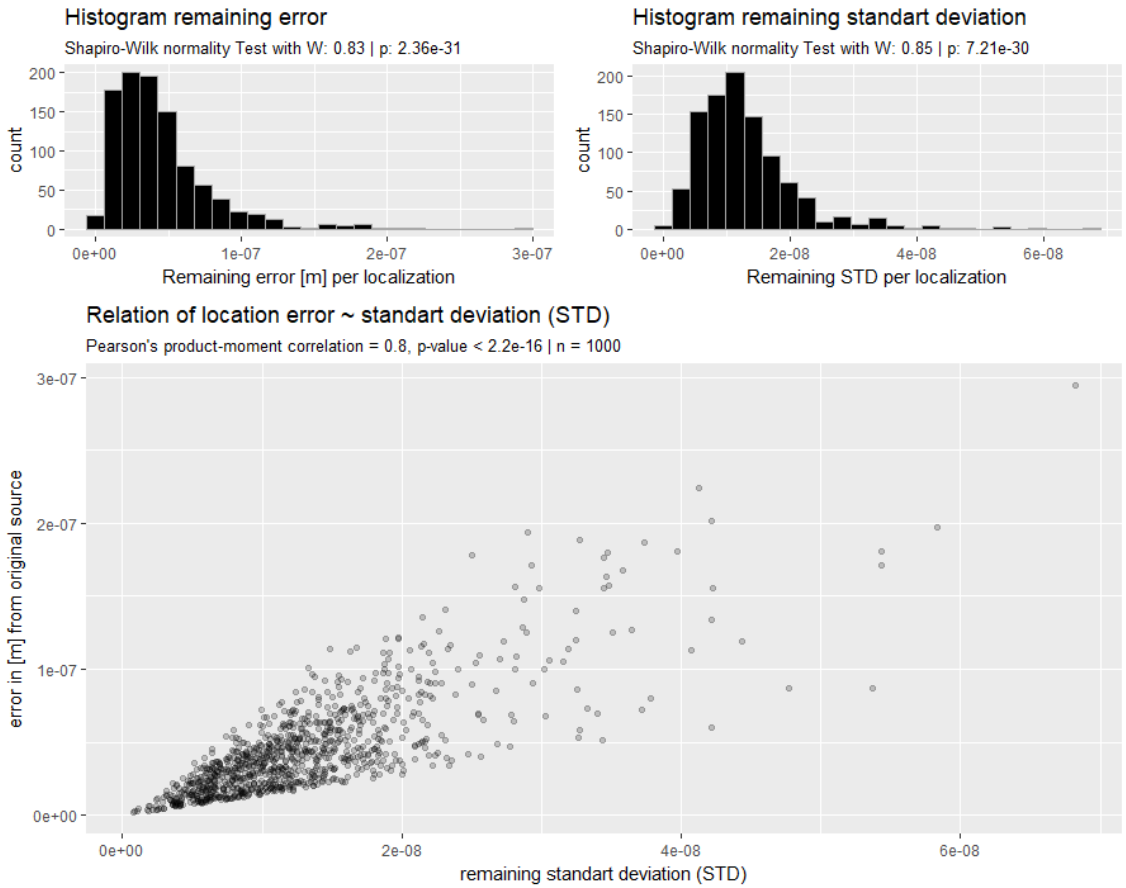

Abbildung 2.5: Statistische Auswertung des auf Standardabweichung (σ) minimierenden Lokalisationsalgorithmus. Es zeigt sich, mit einer Pearson's product-moment correlation von 0.8 (p-value $< 2.2^{-16}$), eine starke Korellation zwischen der verbleibenden Standardabweichung und dem verbleibendem Fehler.

3 | Methodik

Das Kapitel Methodik beschreibt die zur Beantwortung der unter Abschnitt 1.3 auf Seite 3 genannten Forschungsfragen verwendeten Methoden und argumentiert deren Auswahl. Die erfassten Daten sind über *Zenodo* veröffentlicht, siehe HOEDT (2018a,b,c).

3.1 Der Versuchsaufbau

Als Untersuchungsgebiet ist der Innenhof des Campus der Hochschule Ostwestfalen-Lippe (HS OWL) am Standort Höxter definiert. Als exemplarischer Aufbau eines PAMS werden vier Sensoren, mit jeweils einem Mikrofon ausgebracht (siehe Abbildung 3.1 auf der nächsten Seite, Abbildung 3.2 auf der nächsten Seite sowie Abbildung 3.3 auf Seite 14). Die verwendete Sensor-Hard- und Software ist unter Abschnitt 3.2 auf Seite 14 beschrieben.

Die Mikrofon Positionen wurden per Tachymeter (*Leica Builder, Leica Camera AG, Wetzlar, DE*) eingemessen (siehe Abbildung 3.4 auf Seite 15). Im Innenhof (Abbildung 3.5 auf Seite 15) wurde ein Lautsprecher positioniert sowie dessen Wiedergabeposition (siehe Signalpunkte in Abbildung 3.1 auf der nächsten Seite) und Ausrichtung (siehe Richtungsmarker ebenda) aufgenommen. Die für das Aufmaß genutzten Festpunkte lagen in ETRS89/UTM32 (EPSG:25832) mit Höhen über dem GRS80 (EPSG:7019) Ellipsoiden vor. Da die Daten des Laserscan über dem Höhenreferenzsystem DHHN2016 (EPSG:7837) vorlagen, sind die eingemessenen Punkte mittels *TRABBI-2D (Geobasis NRW, Bonn, DE)* unduliert worden.

Als Lautsprecher wird ein *JBL Flip 4 (Harman Deutschland GmbH, Garching b. München, DE)* mit den in Tabelle 3.1 auf Seite 14 gezeigten technischen Daten verwendet.

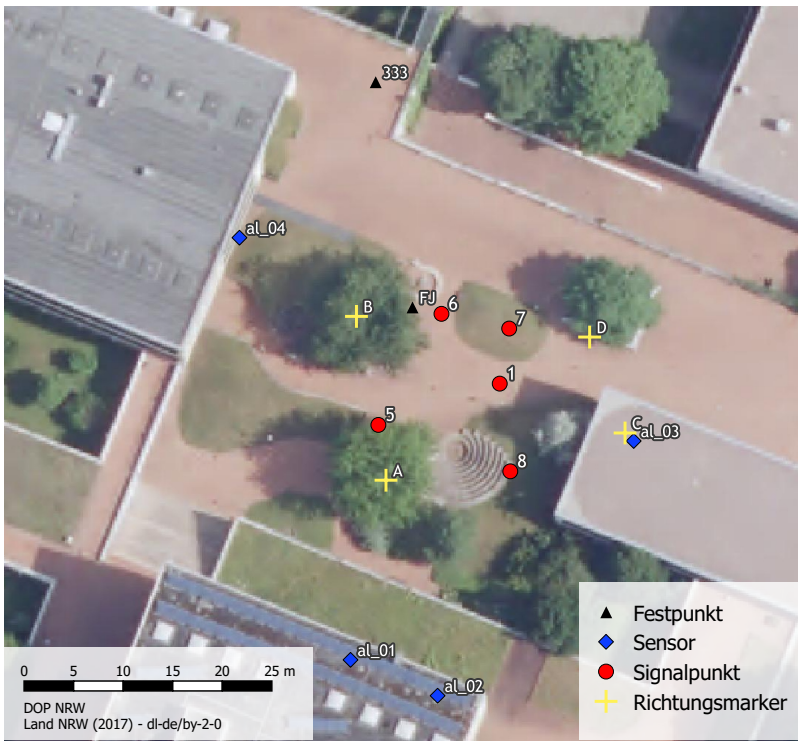

Abbildung 3.1: Karte des Versuchsaufbaus auf dem Innenhof des Campus.

Abbildung 3.2: Der Sensor *al_01* im Gebäude des Campus.

Abbildung 3.3: Das Mikrofon des Sensors *al_01* an der Außenwand des Gebäudes.

Tabelle 3.1: Technische Spezifikation des verwendeten Lautsprechers

Spezifikation	Wert
Signalumwandler	2 x 40 mm
Frequenzabhängigkeit	70 Hz – 20 kHz
Rauschabstand	$\geq$ 80 dB

Der mittlere Schalldruck der abgespielten Audiodatien wurde mittels der *Android* App *Science Journal (Google LLC, Mountain View, US)* erfasst. Er entsprach bei 1 m Abstand im Durchschnitt 35 dB.

3.2 Verwendetes PAMS

3.2.1 Hardware

Da derzeit ein Trend hin zu low-cost PAMS absehbar ist BROWNING ET AL. (2017); WHYTOCK & CHRISTIE (2017) wurde diese These unter Verwendung von *Raspberry Pi 3B (Farnell element14, Leeds, UK)* Microcomputern sowie einfachen und günstigen *Foxnovo Portable USB 2.0-Kondensator-Mikrofon (Typ SF-555B)* realisiert (siehe Tabelle 3.2 auf Seite 16).

Abbildung 3.4: Verwendeter Tachymeter der Firma Leica.

Abbildung 3.5: Innenhof des Campus der HS OWL Höxter.

Tabelle 3.2: Für das PAMS verwendete Komponenten eines Sensors sowie deren Kosten

Komponente	Kosten
Raspberry 3B	34,99 €
Gehäuse	7,99 €
Micro-SDHC Card 32 GB Class 10	16,99 €
USB-Mikrophon	7,99 €
Gesamtkosten	**67,96 €**

3.2.2 Software

Die Sensoren nutzen als Operation System (OS) *Debian Stretch*, welches wie in Abschnitt A.2 auf Seite 60 dokumentiert aufgebaut und als Image auf die restlichen Sensoren verteilt wurde. Die Aufnahmelogik wurde in *Python* programmiert (Quellcode siehe Abschnitt A.3 auf Seite 62). Die Aufnahme wird mittels eines Skriptes konfiguriert (siehe Listing A.2) und über das *pyalsaaudio* Package (IMMISCH & WILSTRUP, 2017) als WAV-Datei gespeichert. Hierbei wird die OS Zeit, wie in Listing A.3 Zeile 58ff dokumentiert, als Zeitstempel der Aufnahme ausgelesen und als Teil des Dateinamens verwendet. Der Zeitstempel wird entsprechend der Standartvorgabe des R-Packages *monitoR* (KATZ ET AL., 2016b) formatiert und für die Lokalisation (siehe Abschnitt 3.6 auf Seite 24) um die Angabe der Microsekunden ergänzt. Die verwendete sampling rate der Aufnahme wurde auf 44.1 kHz gesetzt, wie von BROWNING ET AL. (2017); AIDE ET AL. (2013); FROMMOLT & TAUCHERT (2014) für avifaunistische Aufnahmen verwendet und empfohlen. Mittels Network Time Protocoll (NTP) (MILLS, 1991) wird die Sensor OS Zeit über das Hochschulnetzwerk synchronisiert.

3.3 Klassifikation der Audiodaten

Für die Klassifikation der aufgenommenen Daten wird der vergleichsweise einfache spectrogramm cross correlation Ansatz des *R* (R FOUNDATION, 2018) package *monitoR* (KATZ ET AL., 2016b) genutzt. Der Algorithmus liegt anwendungsbereit vor und hat gute Klassifikationsergebnisse erzielt (KATZ ET AL., 2016a).

Tabelle 3.3: Die Auswahl der Audiodateien für die Erstellung der templates (Trainings-Datensatz)

Bundesland	Region	Geschlecht	Soundqualität	Dateiname
Sachsen	Leipziger Tieflandsbucht	—	b	Carduelis_carduelis_PF00665
Sachsen	Leipziger Tieflandsbucht	—	b	Carduelis_carduelis_PF00666
Sachsen-Anhalt	—	—	b	Carduelis_carduelis_PF00661
Sachsen-Anhalt	—	—	b	Carduelis_carduelis_PF00662
Bayern	—	male	a	Carduelis_carduelis_Tre_P0105_01
Nordrhein-Westfalen	Senne	male	a	Alauda_arvensis_Co0014_10
Nordrhein-Westfalen	Senne	male	a	Alauda_arvensis_Co0021_11
Brandenburg	Naturpark Barnim	male	b	Alauda_arvensis_DIG0071_06
Brandenburg	—	male	a	Alauda_arvensis_V1519_11
Brandenburg	Gamengrund	male	a	Phylloscopus_collybita_DIG0048_05
Berlin	—	male	b	Phylloscopus_collybita_V1692_18

Bei den genutzten Vogellauten wurde die Auswahl auf im Untersuchungsgebiet vorkommende Arten (*Alauda arvensis*, *Carduelis carduelis*, *Phylloscopus collybita*) gesetzt, sowie auf eine unterschiedliche Gesangsstruktur geachtet. Während *Phylloscopus collybita* einen sehr monotonen und gleichförmigen Gesang besitzt, haben die beiden anderen Arten diverse und längere Strophen, wobei *Carduelis carduelis* die diversisten Strophen ausbildet. Ein weiteres Argument für die Auswahl von *Phylloscopus collybita* war die Vergleichbarkeit mit anderen Untersuchungen, wie zum Beispiel PTACEK ET AL. (2016). Die Audiodaten wurden aus dem Tierstimmenarchivs des Museums für Naturkunde Berlin bezogen (MUSEUM FÜR NATURKUNDE BERLIN, 2017).

Für die verwendeten Audiodaten zwischen Training- und Test-Datensätzen wurde auf eine übereinstimmende Regionalität geachtet, wie in den Tabellen 3.3 sowie 3.4 aufgezeigt.

Die jeweiligen Training-Datensätze aus Tabelle 3.3 wurden über *Adobe Audition CS 5.5 (Adobe Systems Software Ireland Limited, Dublin, Republic of Ireland)* betrachtet und auf charakteritische Segmente im Spektrogramm zugeschnitten. Die corellation templates wurden über die *makeCorTemplate()* Funktion des *monitoR* package mit einem score cutoff von 0.4, sowie einem Hanning Window mit Länge 512 pt und einem overlap von 0 % erstellt, wodurch eine zeitliche Auflösung von $\sim$0.012 Sekunden je Bin erreicht wird. Die Definition der im template enthaltenen Frequenz-/Zeit-Bereiche wurde weiterhin über die in *monitoR* implementierte

Tabelle 3.4: Die Auswahl der über den Lautsprecher abgespielten Audiodateien (Test-Datensatz)

Bundesland	Region	Geschlecht	Soundqualität	Dateiname
Sachsen-Anhalt	—	—	b	Carduelis_carduelis_PF00639
Sachsen	Leipziger Tieflandsbucht	—	b	Carduelis_carduelis_PF00643
Sachsen	Leipziger Tieflandsbucht	—	b	Carduelis_carduelis_PF00645
Bayern	—	—	b	Carduelis_carduelis_PF00646
Nordrhein-Westfalen	Lipper Bergland	male	b	Alauda_arvensis_Co0045_11
Nordrhein-Westfalen	Senne	—	b	Alauda_arvensis_Co0075_01
Brandenburg	Naturpark Barnim	—	b	Alauda_arvensis_DIG0053_13
Brandenburg	—	male	b	Phylloscopus_collybita_DIG0069_15
Berlin	Naturpark Barnim	male	b	Phylloscopus_collybita_DIG0008_05

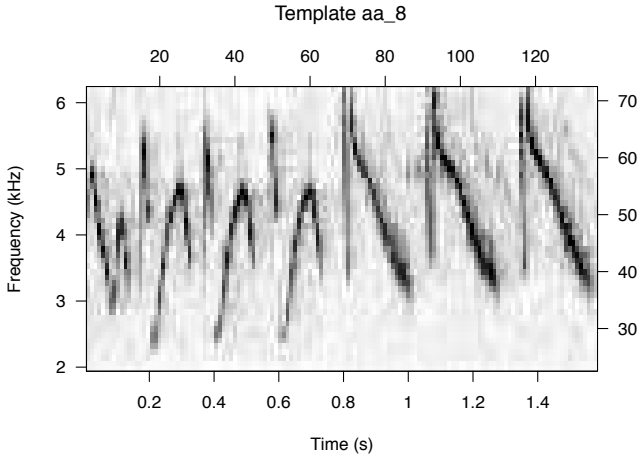

Abbildung 3.6: Über monitoR erstelltes corellation template für *Alauda arvensis* (aa).

Rechteck Auswahl durchgeführt. Beispiele der templates sind in den Abbildungen 3.6, 3.7, sowie 3.8 gezeigt. Insgesamt wurden neun *Alauda Arvensis*, elf *Carduelis carduelis* sowie sieben *Phylloscopus collybita* templates erstellt.

Die so ertellten templates wurden über die *monitoR* Funktionen *corMatch()*, *findPeaks()* und *getDetections()* genutzt, um in den aufgenommenen Audiodateien zeitlich verortet zu werden, wie beispielhaft in Abbildung 3.9 und 3.10, sowie Tabelle 3.5 gezeigt. Der dafür verwendete R-Code ist unter Abschnitt A.5 auf Seite 75 gelistet. Die so detektierten templates wurden je Sensor und Art als CSV-Datei gespeichert (siehe Zeile 119 in Listing A.5). Hierbei wird die verwendete Audiodatei des Sensors, der Zeitstempel der Detektion und die dazugehörige detection score (siehe Abschnitt 2.2 auf Seite 5 - Spektrogramm Corellation) gelistet.

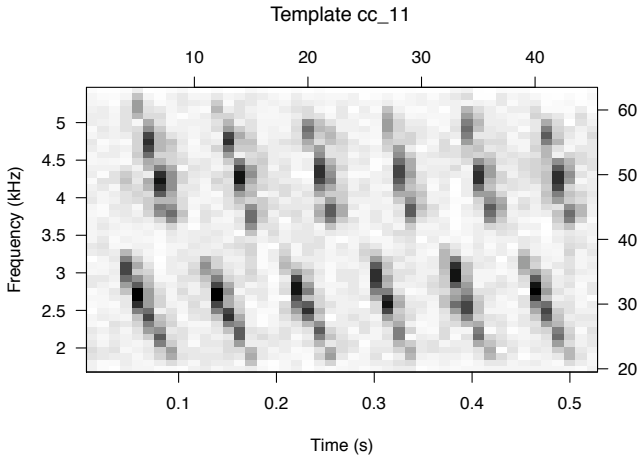

Abbildung 3.7: Über monitoR erstelltes corellation template für *Carduelis carduelis* (cc) – Spektrogramm mit Hanning Window length 512 pt und 0 % overlap.

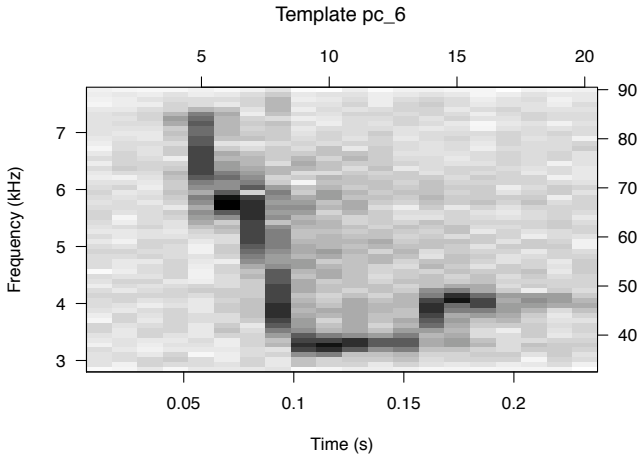

Abbildung 3.8: Über monitoR erstelltes corellation template für *Phylloscopus collybita* (pc) – Spektrogramm mit Hanning Window length 512 pt und 0 % overlap.

Tabelle 3.5: Beispiel eines der erkannten templates (*Alauda arvensis* aa_2) mit den dazugehörigen Audiodateien der Sensoren. Die *relative time* ist dabei in Sekunden vom Start der Audiodatei codiert. Die Spektrogramme der Dateien mit eingezeichneten Zeitstempeln sind in Abbildung 3.10 auf Seite 21 gezeigt.

sensed template	matched File	relative time
aa_2	al_01_2017-10-25_152232_UTC–206883.wav	4.260861678
aa_2	al_02_2017-10-25_152233_UTC–615016.wav	2.890884354
aa_2	al_03_2017-10-25_152234_UTC–040995.wav	2.380045351
aa_2	al_04_2017-10-25_152234_UTC–392631.wav	2.078185941

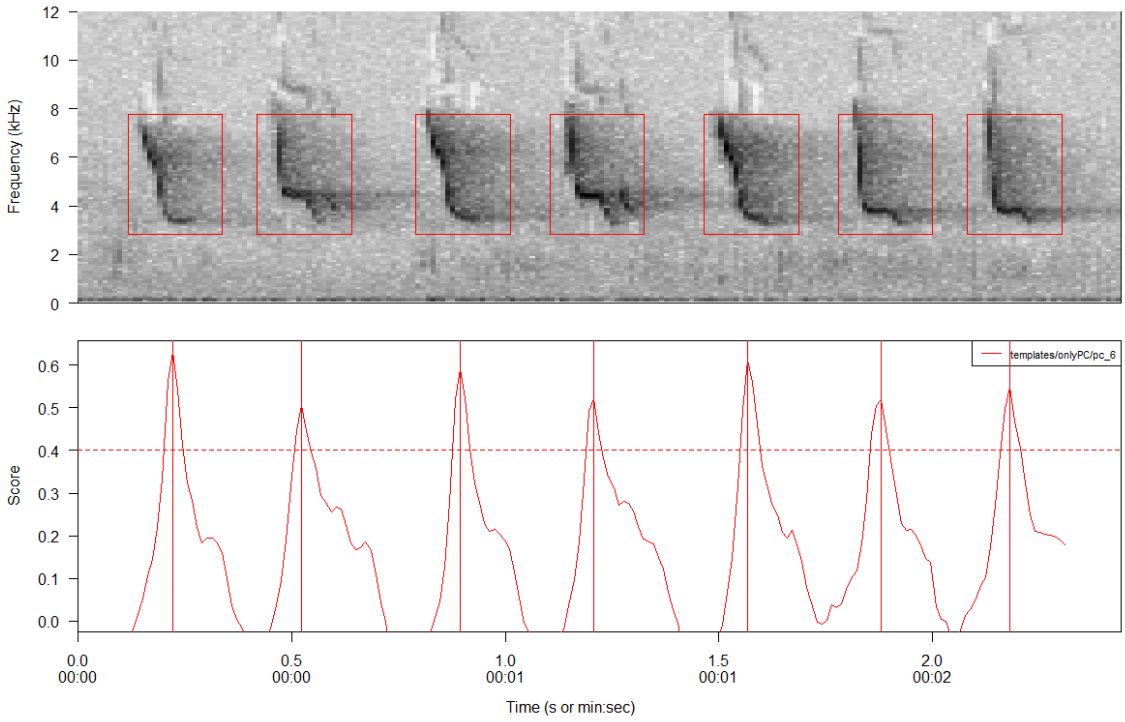

Abbildung 3.9: Über monitoR erstellte cross corellation für das in Abbildung 3.8 auf der vorherigen Seite gezeigte template über einen *Phylloscopus collybita* Gesang – Spektrogramm mit Hanning Window length 512 pt und 0 % overlap.

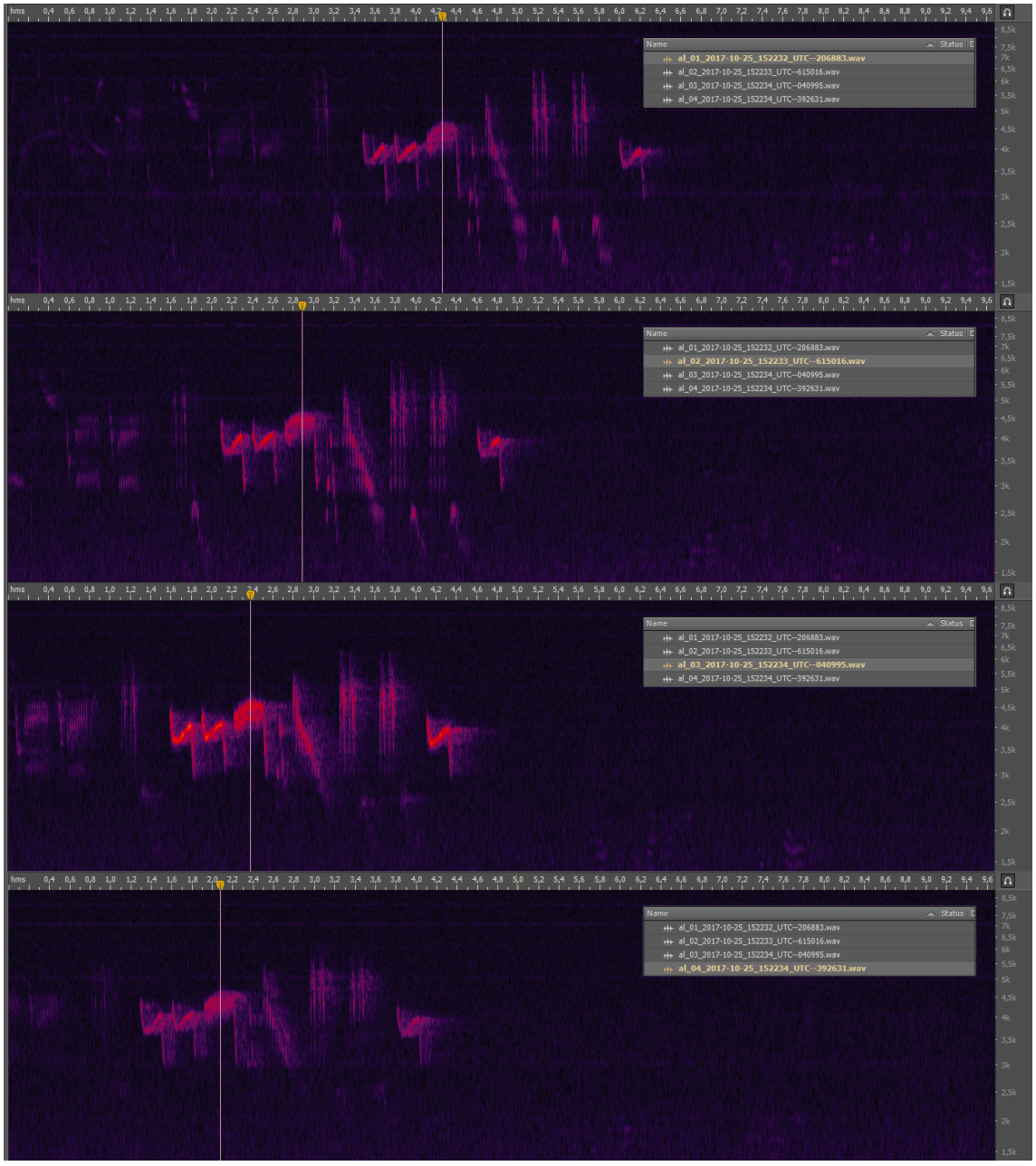

Abbildung 3.10: Beispiel von Zeitstempeln der in Tabelle 3.5 auf Seite 19 gezeigten Detektion in den jeweiligen Audiodateien. Die Spektrogrammansicht wurde mittels *Adobe Audition CS 5.5 (Adobe Systems Software Ireland Limited, Dublin, Republic of Ireland)* angefertigt.

Tabelle 3.6: Gekürzte Fassung der während der Aufnahme abgespielten Audiodateien. Der Zeitstempel ist in hhmmss codiert. Die gesamte Tabelle (A.2) ist dem Anhang zu entnehmen.

Time	Point	Direction	Species
152223	1	A	aa
152232	1	A	aa
152340	1	A	aa
152409	1	A	cc
...	...	...	...
164246	8	D	cc
164251	8	D	pc
164256	8	D	pc
164320	8	D	pc

Während der Datenaufnahme sind die Zeitstempel der jeweils abgespielten Audiodaten nach Spezies dokumentiert worden, wie beispielhaft in Tabelle 3.6 dargestellt.

3.4 Ermittlung des Azimuth

Um die in Abschnitt 1.3 auf Seite 3 genannte These 2.c zu prüfen, wird die Ausrichtung des Lautsprechers im Verhältnis zu den Sensorpositionen ermittelt. Hierbei wurde vom jeweiligen Signalpunkt der Lautsprecher auf einen Richtungsmarker ausgerichtet, wie beispielhaft in Abbildung 3.11 auf der nächsten Seite dargestellt. Für die Berechnung des Azimuth wurde die PostGIS Funktion *ST_Azimuth*, wie in Abschnitt A.6 auf Seite 79 beschrieben, genutzt. Da *ST_Azimuth* den rechtswinkligen Azimuth von Nord ausgibt, ist das Winkelmaß des jeweiligen $\Delta^{\measuredangle}$ zwischen Sensor und Richtungsmarker zu berechnen. Gegeben seien die nordgerichteten rechtswinkligen $\measuredangle_R$ zum Richtungsmarker, sowie der ebenso angegebene $\measuredangle_S$ zum betrachteten Sensor. Für den $\Delta^{\measuredangle}_{RS}$ gilt Gleichung (3.1).

$$\Delta^{\measuredangle}_{RS} = |\measuredangle_R - \measuredangle_S| \tag{3.1}$$

Die Berechnung der Gleichung (3.1) wurde über Excel durchgeführt. Die Summe

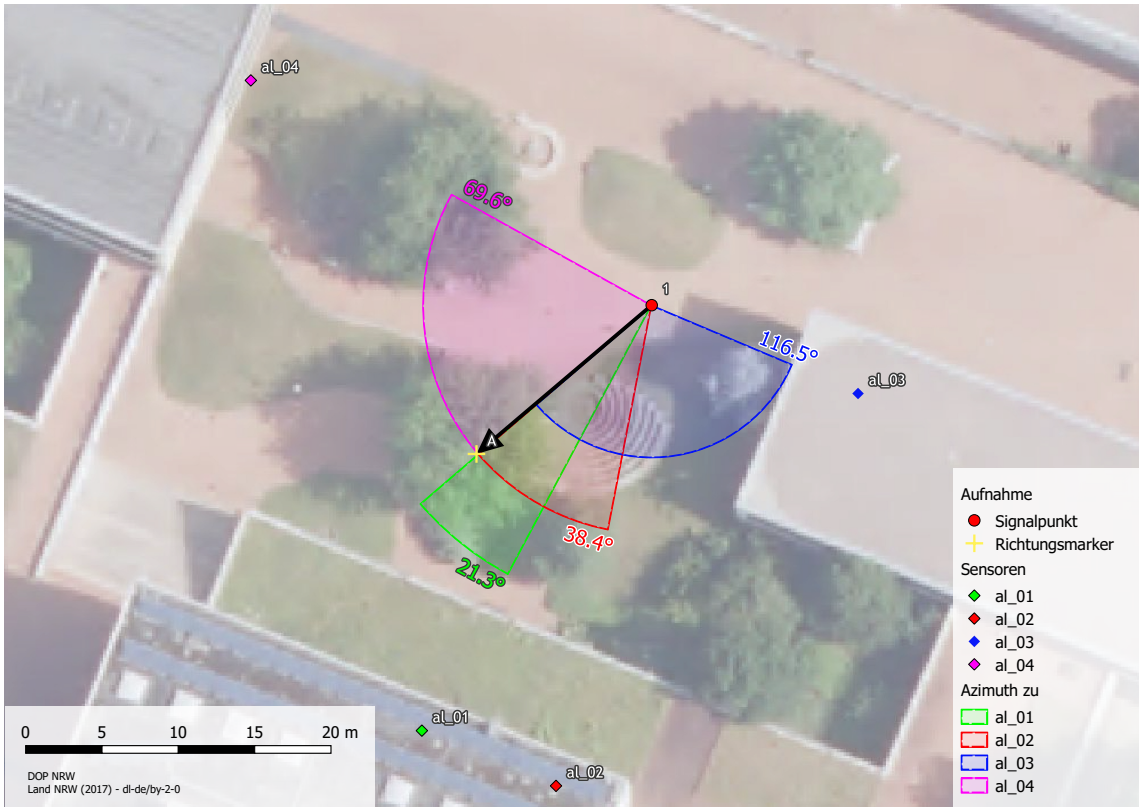

Abbildung 3.11: Berechnung des summierten Azimuth je Signalposition - Richtungsmarker Paar (hier Signalposition 1 und Richtung A).

der so ermittelten $\Delta^{\measuredangle}_{RS}$ wird als Metrik der Ausrichtung des Lautes genutzt.

Die Ergebnisse der Berechnung sind über den *Python Pandas* Code des Listing A.8 in Zeile 424 mit den jeweiligen Datensätze verbunden (inner join) worden.

3.5 Ermittlung Schallverschattung

Für die Berechnung der Schallverschattung werden die Daten des Laserscans NRW LAND NRW (2017) verwendet. Um die in Abschnitt 1.3 auf Seite 3 genannte These 2.a zu prüfen wird von der Schallgeberposition eine 3D-Röhre zu den jeweiligen Mikrofonen konstruiert und ein 3D-Intersect mit der Punktwolke des Laserscans durchgeführt. Hierbei entspricht die Anzahl der überschnittenen Punkte dem Maß der Schallverschattung. Der verwendete PostGIS Code ist unter Abschnitt A.7 auf Seite 81 aufgezeigt. Es wurden verschiedene Durchmesser der Röhre geprüft und letztlich als Durchmesser der 3D-Röhre 0.5 m festgesetzt. Bei höheren Wer-

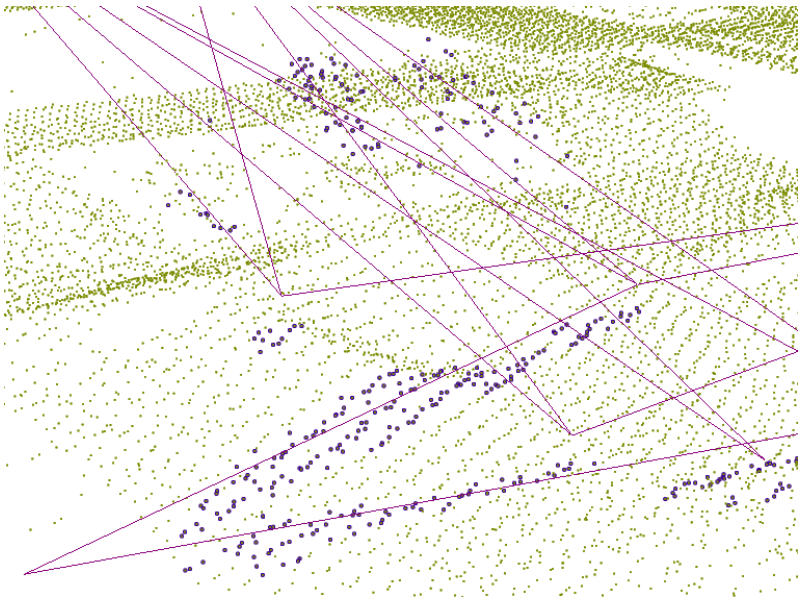

Abbildung 3.12: Abfrage der LIDAR Returns nach Schallkegel für 1 m Radius. Es werden viele Bodenpunkte ausgewählt. Die verwendete Abfrage basiert daher auf 0.5 m Radius.

ten sind zu viele irrelevante Bodenpunkte selektiert worden, wie beispiehaft in Abbildung 3.12 dargestellt.

Die Ergebnisse der Berechnung sind über den *Python Pandas* Code des Listing A.8 in Zeile 428f mit den jeweiligen Datensätzen verbunden worden.

3.6 Lokalisierung

Das unter Abschnitt 3.2 auf Seite 14 beschriebene PAMS verwendet zur Lokalisation unter anderem das *R 3.4.4 (R Foundation for Statistical Computing, Vienna, Austria,* R CORE TEAM *(2018))* package *Sound Finder* (WILSON ET AL., 2014). Da *Sound Finder* mit den teilweise fehlerhaften und nicht ausreichend genau synchronisierten Aufnahmen keine Lokalisation vornehmen konnte (siehe Abschnitt 5.1.2 auf Seite 44), wurde der in Abschnitt 2.3 auf Seite 7 beschriebene, auf dem *Scipy.Minimize* basierende Lokalisierungsalgorithmus zur Datenauswertung angewandt. Der in diesem Abschnitt gezeigte Programmcode ist als vollständiges Listing im Abschnitt A.8 auf Seite 90 zu finden.

Hierbei wird zunächst aus den einzelnen Detektionen des *monitoR* Codes ein *Python Pandas Dataframe* zusammengestellt. Die relativen Zeitstempel der Detektio-

nen werden anhand der im Dateinamen gespeicherten absoluten UTC-Zeitstempel zu absoluten Detektionszeiten verrechnet, wie in Listing 3.1 gezeigt.

Listing 3.1: Berechnung der absoluten Detektionszeiten

```
#%% PROCESS DETECTION FILES
# get file.names datetime from the recordings start
# + relative timedelta of the detection by monitoR in seconds from the samples
# start to get the UTC sensed time of the detection per row
def calcSensedTime(row):
    timeStart = datetime.datetime.strptime(row['file.names[i]'][6:-4],
                                           "%Y-%m-%d_%H%M%S_UTC--%f")
    timeDetection = timeStart + datetime.timedelta(seconds = row['time'])
    return timeDetection

# apply a new row "sensedTime" as function calcSensedTime per row
df_detections['sensedTime'] = df_detections.apply(calcSensedTime, axis=1)

# index the detections by sensed time as timeseries
df_detections_indexed = df_detections.set_index(['sensedTime'])
```

Für eine erfolgreiche dreidimensionale Lokalisation mit der oben genannten Methode sind vier Detektionen desselben Ereignisses notwendig. Da jede Detektion mit der Angabe des erkannten templates und der absoluten Zeit vorliegt, können über die *Pandas* Methoden *iloc* und *get_loc* die jeweils zeitlich nähesten Detektionen desselben templates identifiziert werden (Listing 3.2).

Listing 3.2: Selektion der zeitlich nähesten Detektion

```
for index, row in df_detections_indexed.iterrows():
    index_matchingDetections = index_matchingDetections + 1


    toDo = toDo - 1
    print("todo :: " + str(toDo))

    sensedTime = index
```

```
    # not only search for species (eg. Alauda arvensis)
    # but also for the exact correlation template match!
    # otherwise there are loads of false positives!
    species = row['template']
    sensor = row['sensor']

    # get played species and direction
    nearestPlayed = df_played_indexed.iloc[df_played_indexed.index
        ↪ .get_loc(sensedTime,method='nearest')]

    # get temperature
    nearestTemperature = df_temperatures.iloc[df_temperatures.
        ↪ index.get_loc(sensedTime,method='nearest')]
```

Da die maximale Distanz zwischen den Sensoren des Versuchsaufbaus (Abbildung 3.1 auf Seite 13) ∼50 m beträgt, wurde jede Detektion mit einer höheren Δ-Zeit als 0.15 Sekunden abgelehnt, was der maximalen zeitlichen Abweichung eines Signals innerhalb des Sensornetzwerkes entspräche (Siehe Listing 3.3).

Listing 3.3: Selektion der zeitlich nähesten Detektion

```
        # calculate timedelta between sensed time of master
            ↪ sensor
        # and the nearest detection on one of the slave sensors
        tDelta = sensedTime - nearestDetection['sensedTime']

        if (abs(tDelta) < maxDeltaTime ):
            # add sensors detection as relative timedelta
            # to the detection dict
            detection[sensorName+"_matchedFile"] =
                ↪ nearestDetection['file.names[i]']
            detection[sensorName+"_sinceFileStart"] =
                ↪ nearestDetection['time']
            detection[sensorName+"_detectionScore"] =
                ↪ nearestDetection['score']
            detection[sensorName] = tDelta.total_seconds()
            detection["summed_detectionScore"] = detection["
                ↪ summed_detectionScore"] + nearestDetection['
                ↪ score']
        else:
```

```
14              message = row['file.names[i]'] + " :: deltatime is
                    ↪ to big :: " +str(abs(tDelta)) + " :: detection
                    ↪ will be discarded"
15              errorDetections[index_matchingDetections] =
                    ↪ detection
16              errors.append(message)
17              print(message)
```

Die zur Berechnung der Schallgeschwindigkeit genutzte Formel ist in Gleichung (3.2) gezeigt.

$$v = 331.3 * \sqrt[2]{1 + Temperatur/273.15} \tag{3.2}$$

Die Formel ist wie in Listing A.1 dokumentiert zur Berechnung der TDOA basierten Distanzdifferenzen (siehe grüne Umkreise in Abbildung 2.4 auf Seite 10) verwendet worden. Die hierbei notwendigen Temperaturen in °C wurden, wie in Tabelle A.1 auf Seite 104 gezeigt, an der Wetterstation Höxter ($51°46' 3.36'' N, 9°22' 9,86'' E$, 156m über NN), in unmittelbarer Nähe des Aufnahmeortes erfasst.

Listing 3.4: Berechnung der Distanz aus den Delta Zeitstempeln

```
1  # The speed of sound is calculated from an appropriate
       ↪ temperature value T
2  # by v = 331.3 * sqrt(1 + T / 273.15)
3  def travelledDistance(row,sensor):
4      timedelta = row[sensor]
5      velocity = 331.3 * np.sqrt((1 + row['temperature']/ 273.15))
6      travelledDistance = timedelta * velocity
7      return abs(travelledDistance)
8
9  # apply new row "dn" as function of travelledDistance per row
10 for i in range(1,5):
11     distance = 'd' + str(i)
12     sensor = 'al_0' + str(i)
13     df_matchingDetections_cleaned[distance] =
           ↪ df_matchingDetections_cleaned.apply(travelledDistance,
14                                   args=(sensor,),
15                                   axis=1)
```

Die seperaten Daten der Temperatur wurden über die *Pandas* Methode *iloc* zeitlich mit den Daten der Detektionen (siehe Zeile 19 im Listing 3.2) verbunden. Somit konnte die zur Detektionszeit vorliegende Temperatur in die Gleichung (3.2) auf der vorherigen Seite eingesetzt werden, um die relativen Distanzunterschiede je Aufnahme und Sensor zu berechnen.

Mit den vorliegenden Distanzunterschieden ist die Lokalisierung nach dem *Scipy.Minimize* Algorithmus durchgeführt worden. Der hierfür verwendete Code ist in den Methoden *calcXYZ*, *getLocation* und der dazugehörigen Fitness-Funktion *getSTD* im Listing A.8 ab Zeile 269ff dokumentiert. Als Methode zur Minima Berechnung wurde *Limited-memory BFGS* genutzt, da eine Ausführung des Algorithmus auf Hardware mit begrenzter Speicherkapazität vorstellbar ist, wie zum Beispiel *Raspberry Pis*.

3.7 Verwendete Software

Die statistische Datenauswertung wurde über die open-source Software *R 3.4.4 (R Foundation for Statistical Computing, Vienna, Austria,* R Core Team *(2018))* durchgeführt. Für räumliche Abfragen ist das open-source Geodatenbankmanagementsystem *PostgreSQL 9.6.6, 64-bit (The PostgreSQL Global Development Group)* in Verbindung mit der *PostGIS 2.4.4 (PostGIS Project Steering Committee (PSC))* Erweiterung verwendet worden. Der erstellte Python Code basiert auf Python 3.6.1 und wurde auf dem *Anaconda 4.4.0 (64-bit) Interpreter (Anaconda Inc., Austin TX, US)* ausgeführt.

4 | Ergebnisse

Im Kapitel Ergebnisse werden die erfassten Daten und die Ausgaben der verwendeten Algorithmen in Tabellen und Diagrammen dargestellt und bezüglich der aufgestellten Forschungsfragen statistisch ausgewertet.

4.1 Deskriptive Statistik

Im Folgenden werden die erfassten Datensätze beschrieben und veranschaulicht dargestellt. Hierbei wird unterteilt nach den Ergebnissen der Klassifikation und der Lokalisation.

4.1.1 Klassifikation

Die folgenden Anzahlen und Auswertungen beziehen sich auf die jeweiligen Detektionen der erkannten Ereignisse innerhalb der Aufnahmedaten. Hierbei gilt eine Detektion als valide, wenn:

1. alle vier Sensoren das selbe template
2. innerhalb der festgelegten Δ-Zeit von maximal 0.15 Sekunden

klassifiziert haben. Die so erkannten Detektionen wurden, wie in Abschnitt 3.6 auf Seite 24 beschrieben, als ein Audio-Event zusammengefasst und analysiert.

In den aufgenommenen Audiodateien wurden insgesammt 15.412 Events erfasst, wie in Tabelle 4.1 auf der nächsten Seite zusammengefasst dargestellt. Es zeigt

Tabelle 4.1: Unterschiede der in den Events detektierten und tatsächlich abgespielten Spezies

Species	detektiert	abgespielt
Alauda arvensis	1.202	2.298
Carduelis carduelis	0	1.013
Phylloscopus collybita	14.210	12.101
Gesamtanzahl	**15.412**	**15.412**

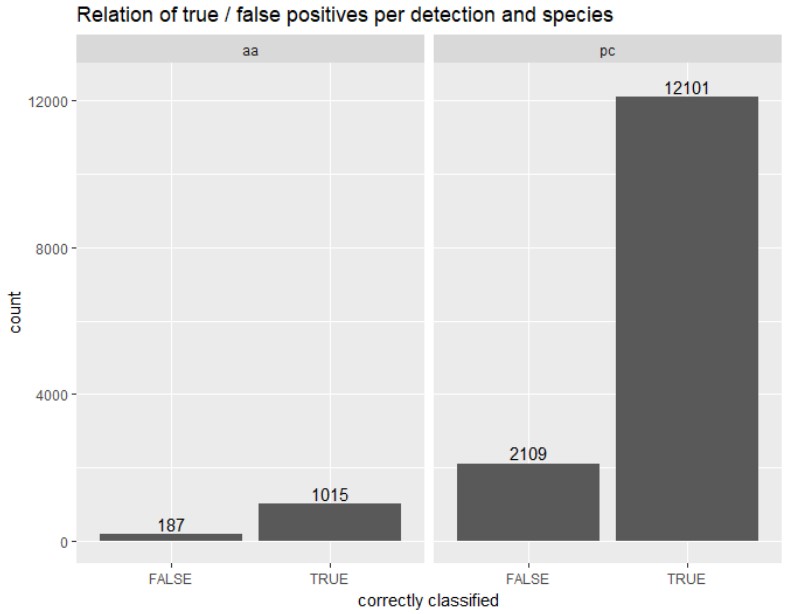

Abbildung 4.1: Verhältnis der wahren und falschen Klassifikationen je detektierter Art. Die Art *Carduelis carduelis* wurde nicht erfasst.

sich, dass keines der *Carduelis carduelis* templates detektiert wurde, sondern die abgespielten Audiodateien der Art als *Phylloscopus collybita*, oder *Alauda arvensis* erkannt wurden. Die Abbildungen 4.1 und 4.2, sowie die Tabelle 4.2 zeigen hierbei, dass die *Phylloscopus collybita* Detektionen zu ∼14.8 % und die *Alauda arvensis* zu ∼15.5 % falsch klassifiziert worden sind. Die *Carduelis carduelis* Audiodateien sind fälschlicherweise zu ∼81.5 % als *Phylloscopus collybita* und zu ∼18.5 % als *Alauda arvensis* klassifiziert worden. *Phylloscopus collybita* zeigt ein hohes Maß an false positives (∼ 17 %).

Wie in Abbildung 4.3 auf Seite 32 dargestellt, zeigen sich große Unterschiede zwischen der Anzahl der detektierten Events und der Aufnahmeposition / Ab-

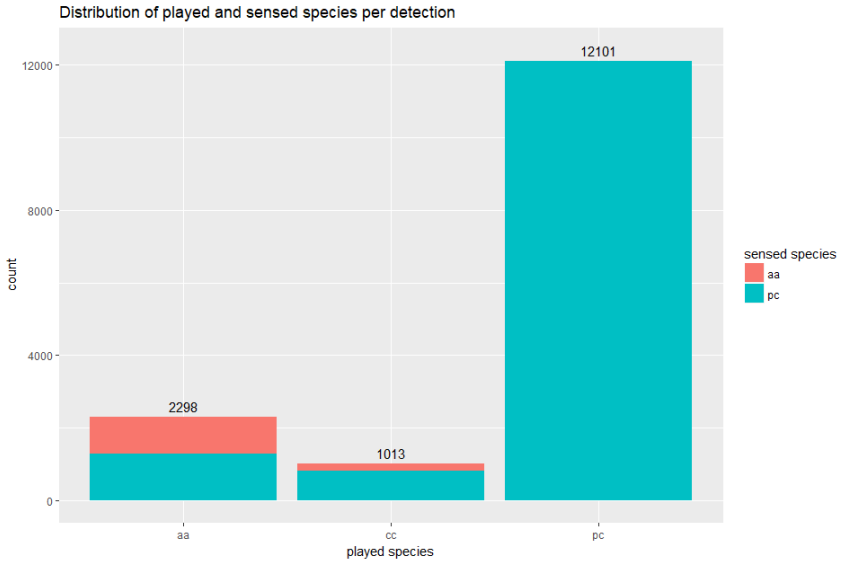

Abbildung 4.2: Unterschiede zwischen abgespielter und klassifizierter Art.

Tabelle 4.2: Verhältnis zwischen true und false positives innerhalb der klassifizierten Detektionen

Typ	aa	cc	pc
true positive	1.015	0	12.101
false positive	1.283	0	2.109
false negative	1.283	1.013	0
true negative	NA	NA	NA

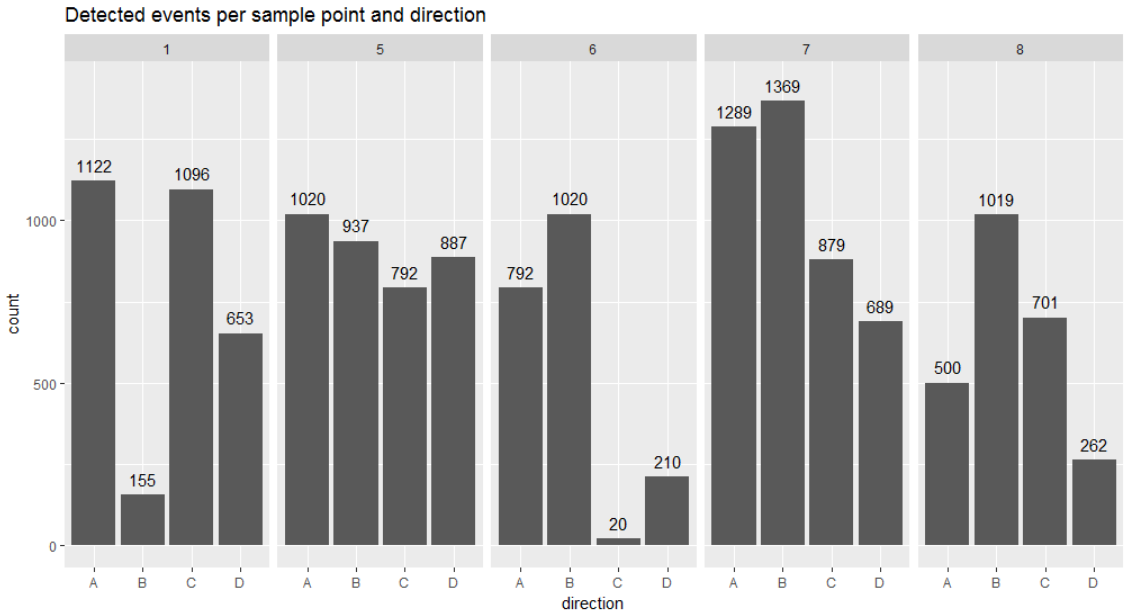

Abbildung 4.3: Anzahl der detektierten Events je Aufnahmepunkt und summierter Abspielrichtung. Je Aufnahmepunkt sind die Audiodateien in vier Richtungen abgespielt worden, die berechneten Δ-Azimuth sind in Abbildung 4.4 dargestellt.

spielrichtung. Die hierbei berechneten Δ-Azimuth Werte sind in Abbildung 4.4 dargestellt. Die größte Anzahl an Events ist an Abspielposition 7 in Richtung des Markers B detektiert worden, die niedrigste Anzahl an Abspielposition 6 in Richtung des Markers C. Abbildung 4.4 auf der nächsten Seite zeigt hierbei die jeweiligen berechneten summierten Δ-Azimuth Werte der Aufnahmeposition / Richtungsmarker Paare.

4.1.2 Lokalisation

Zur Lokalisation und Berechnung der Gleichung (3.2) auf Seite 27 ist die Temperatur zur Aufnahme zu ermitteln. Zur Aufnahmezeit wurden Temperaturen zwischen 14.0–14.7 °C gemessen, wie in Abbildung 4.5 auf der nächsten Seite dargestellt.

Die Lokalisation zeigt Fehlerwerte von $\bar{x}$ 34.96 $\pm$ 19.49 m (min: 1.1, max: 95.8 m), wie in Tabelle 4.3 gelistet. Abbildung 4.7 auf Seite 36 zeigt den Lokalisierungsfehler zwischen den erkannten Arten, den abgespielten Standorten sowie der Tageszeit. Es zeigen sich zu jeder Tageszeit stark streuende Werte für den Lokalisierungsfehler. Diese zeigen Häufungen bei 10 und 45 m Fehler zu unterschiedlicher Abspiel-

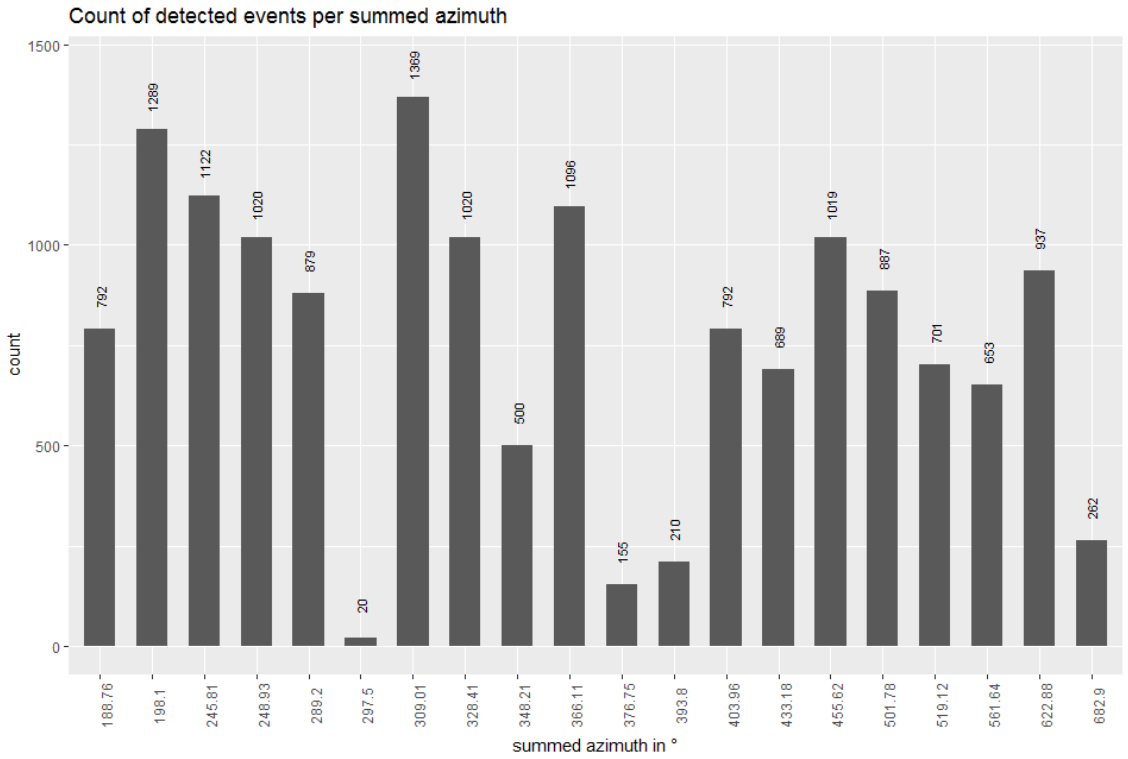

Abbildung 4.4: Anzahl der detektierten Events je Aufnahmepunkt und summiertem Δ-Azimuth in °. Je Aufnahmepunkt sind die Audiodateien in vier Richtungen abgespielt worden.

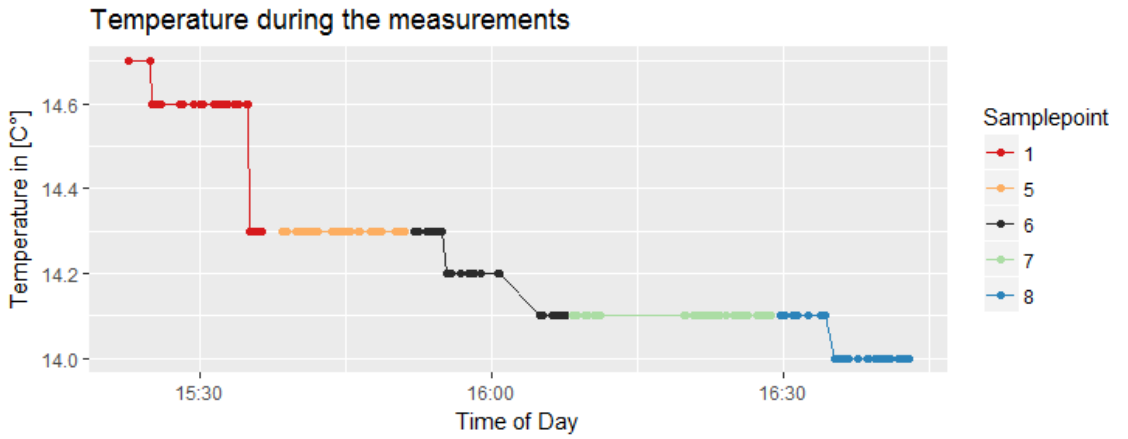

Abbildung 4.5: Gemessene Temperaturen zur Aufnahmezeit

Tabelle 4.3: statistische Maße des Lokalisierungsfehlers.

Minimum	0.25 Quantil	Mittelwert	0.75 Quantil	Maximum	Standardabweichung
1.1	14.24	34.96	49.55	95.82	19.49

Tabelle 4.4: statistische Maße des Lokalisierungsfehlers unterteilt nach abgespielter Vogelart.

abgespielte Vogelart	Minimum	Mean	Maximum	Standardabweichung
aa	2.54	37.55	87.33	18.61
cc	2.96	36.44	95.82	18.26
pc	1.10	34.34	81.22	19.70

zeit und -position. Für die Aufnahmepositionen 6 und 7 zeigt sich, unabhängig der Detektierten Vogelart, ein leicht erhöhter Verortungsfehler.

Die statistischen Maße, unterteilt nach Vogelart, sind in Tabelle 4.4 dargestellt. Die Maße für Standardabweichung sowie arithmetisches Mittel sind unter den Populationen ähnlich. Die Verteilung der Lokalisierungsfehlerwerte innerhalb der Vogelarten ist in Abbildung 4.6 auf der nächsten Seite gezeigt. Der Density-Plot zeigt eine nicht normale, mehrgipflige Verteilung für jede der untersuchten Arten.

In Abbildung 4.8 auf Seite 37 wird die Streuung des Lokalisierungsfehlers nach Abspielort, -Richtung sowie abgespielter Vogelart als Boxplot-Grid dargestellt. Neben zwei nicht vorhandenen Abspielrichtung-Station-Template Paaren, B-1-aa und C-6-pc, zeigen sich für die Stationen 5 und 6 vergleichsweise niedrige Median Lokalisierungsfehler. Die Kombination D-5 zeigt bei den Templates cc und pc die geringsten Fehler. Die größten Fehler zeigen sich in der Kombination C-1 für alle Templates.

Die verorteten Events sind in Abbildung 4.9 auf Seite 38 als Karte gezeigt. Es zeigt sich eine leicht Süd-West zu Nord-Ost gedehnte Streuung sowie eine Häufung an den Grenzen des Algorithmus (Bounds der Fit-Funktion, A.8 Zeile 313). In Abbildung 4.10 sind die Lokalisationen unterteilt nach Abspielposition und Richtung dargestellt. Hierbei zeigt sich ein unterschiedliches Streuungsverhalten nach Position und Richtung. Mit Außnahme von C-6 zeigen sich in jeder Konstellation Cluster von Verortungen. Während die Verortungen der Signalposition 7 einer Süd-West zu Nord-Ost Streuung unterliegen, zeigt sich für die Signal-

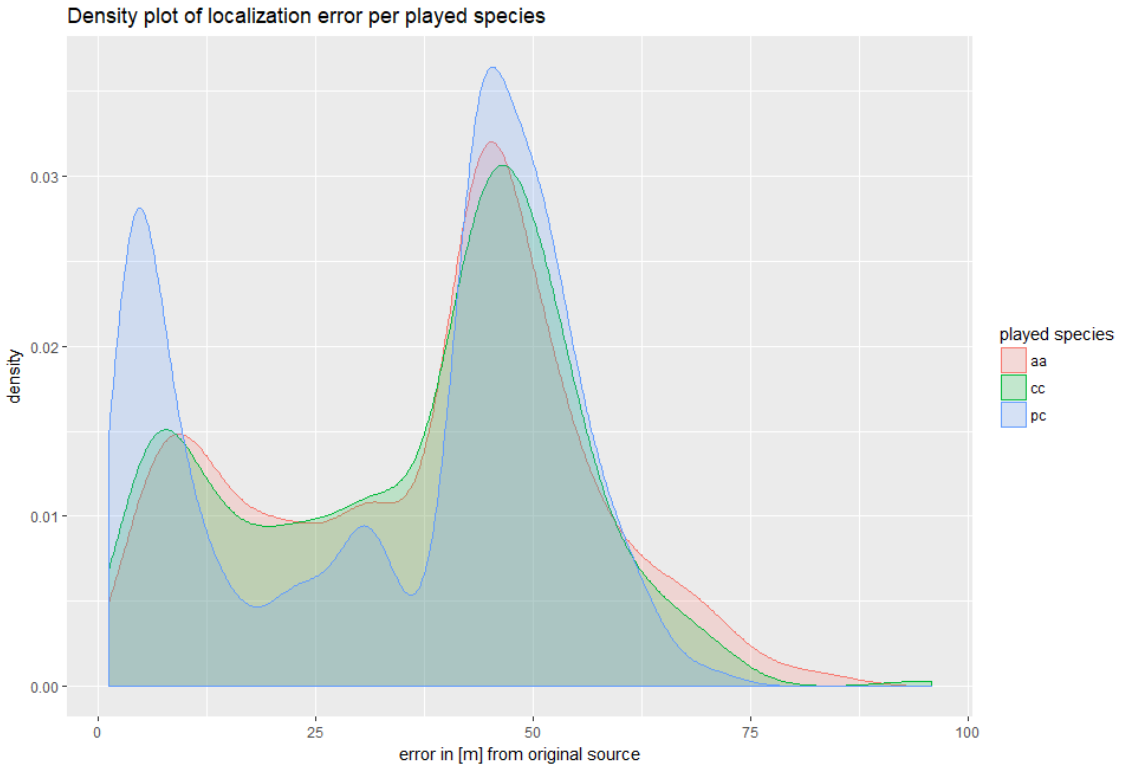

Abbildung 4.6: Verteilung des Lokalisierungsfehlers innerhalb der abgespielten Vogelarten.

position 8 eine Ost-West Streuung. Die Position 5 zeigt die geringste gerichtete Streuung. Die Abbildungen zeigen 3D Datensätze als 2D Ansicht, wodurch die Lokalisierungsfehler kleiner erscheinen als in 3D gemessen.

4.2 Analyse

Im Folgenden werden die zuvor dargestellten Daten fragestellungsbezogen statistisch ausgewertet:

4.2.1 Einfluß der Klassifikationsgüte

Die Hypothese *Die Güte der Klassifikation korreliert positiv mit der Güte der Lokalisation* kann widerlegt werden. In Abbildung 4.11 auf Seite 40 zeigt sich, mit einer Pearson's product-moment correlation von ∼-0.016 und bei einem p-value von 0.042, dass die Nullhypothese H_0 *es liegt keine Korrelation vor* bei einem α-Fehler

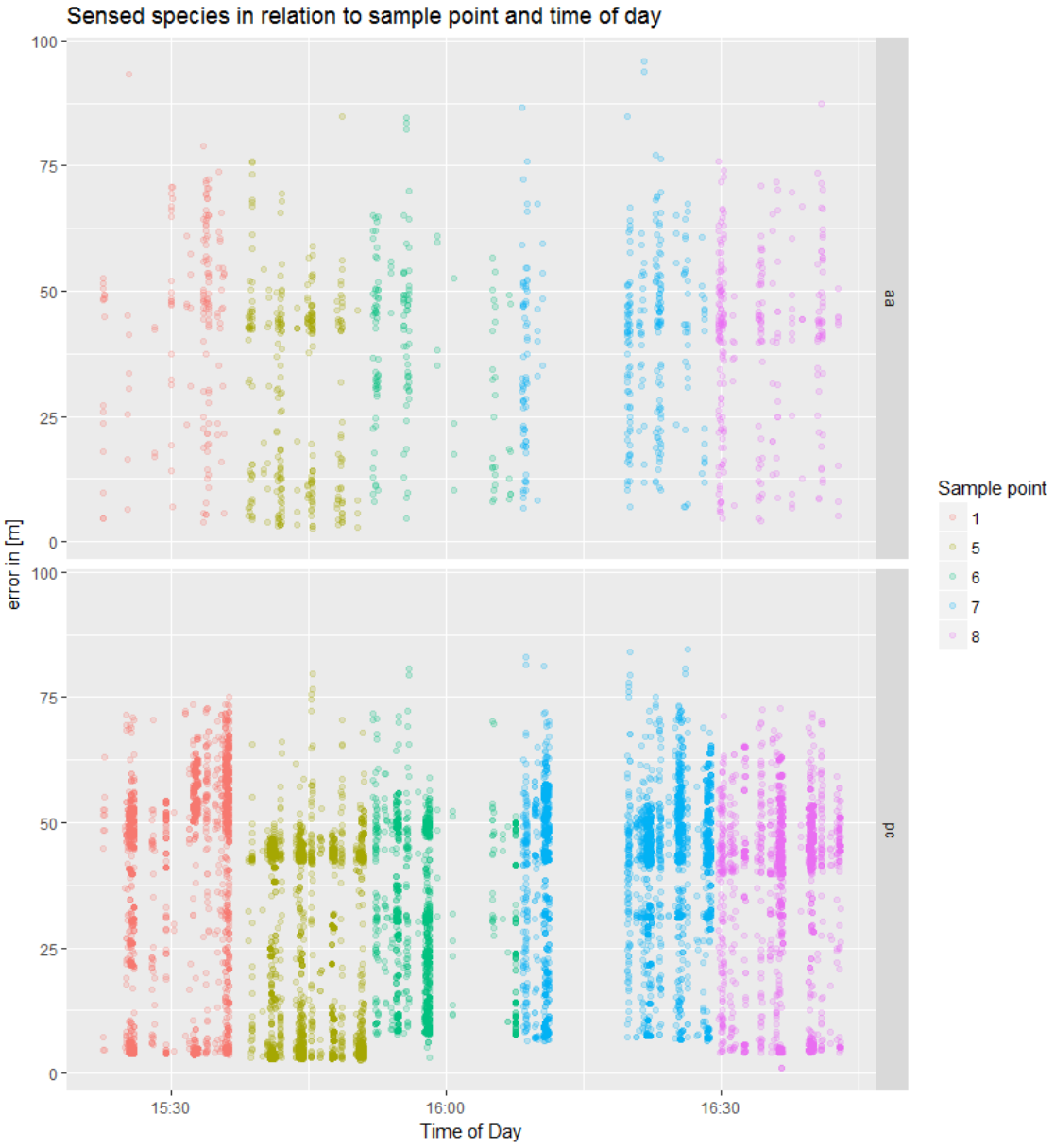

Abbildung 4.7: Erkannte Spezies im Verhältnis zum Aufnahmepunkt, der Tageszeit und dem Lokalisierungsfehler.

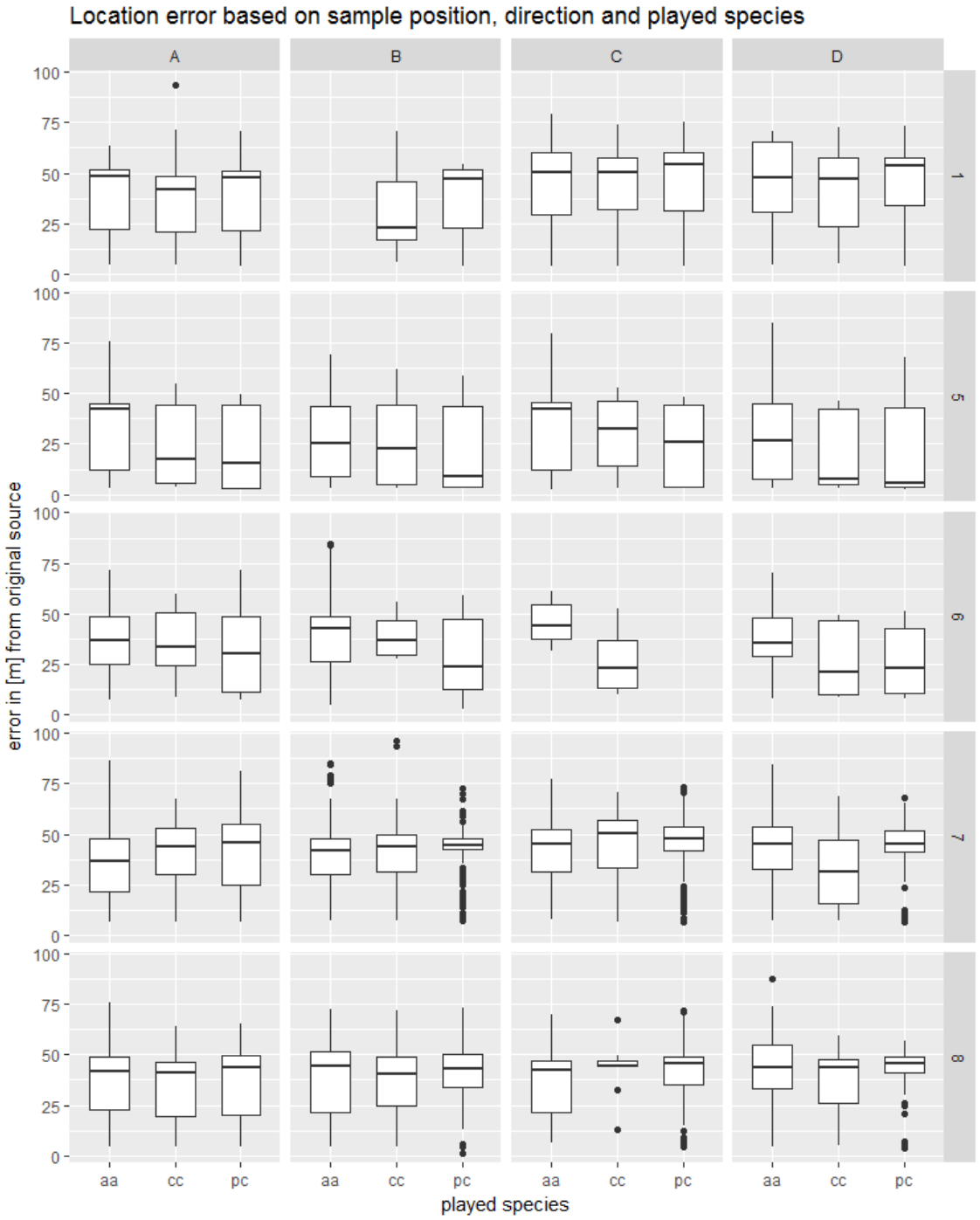

Abbildung 4.8: Lokalisierungsfehler aufgeteilt nach Aufnahmepunkt, Abspielrichtung sowie abgespielter Spezies. Neben zwei nicht vorhandenen Abspielrichtung-Station-Template Paaren, B-1-aa und C-6-pc, zeigen sich für die Stationen 5 und 6 vergleichsweise niedrige Median Lokalisierungsfehler. Die Kombination D-5 zeigt bei den Templates cc und pc die geringsten Fehler. Die größten Fehler zeigen sich in der Kombination C-1 für alle Templates

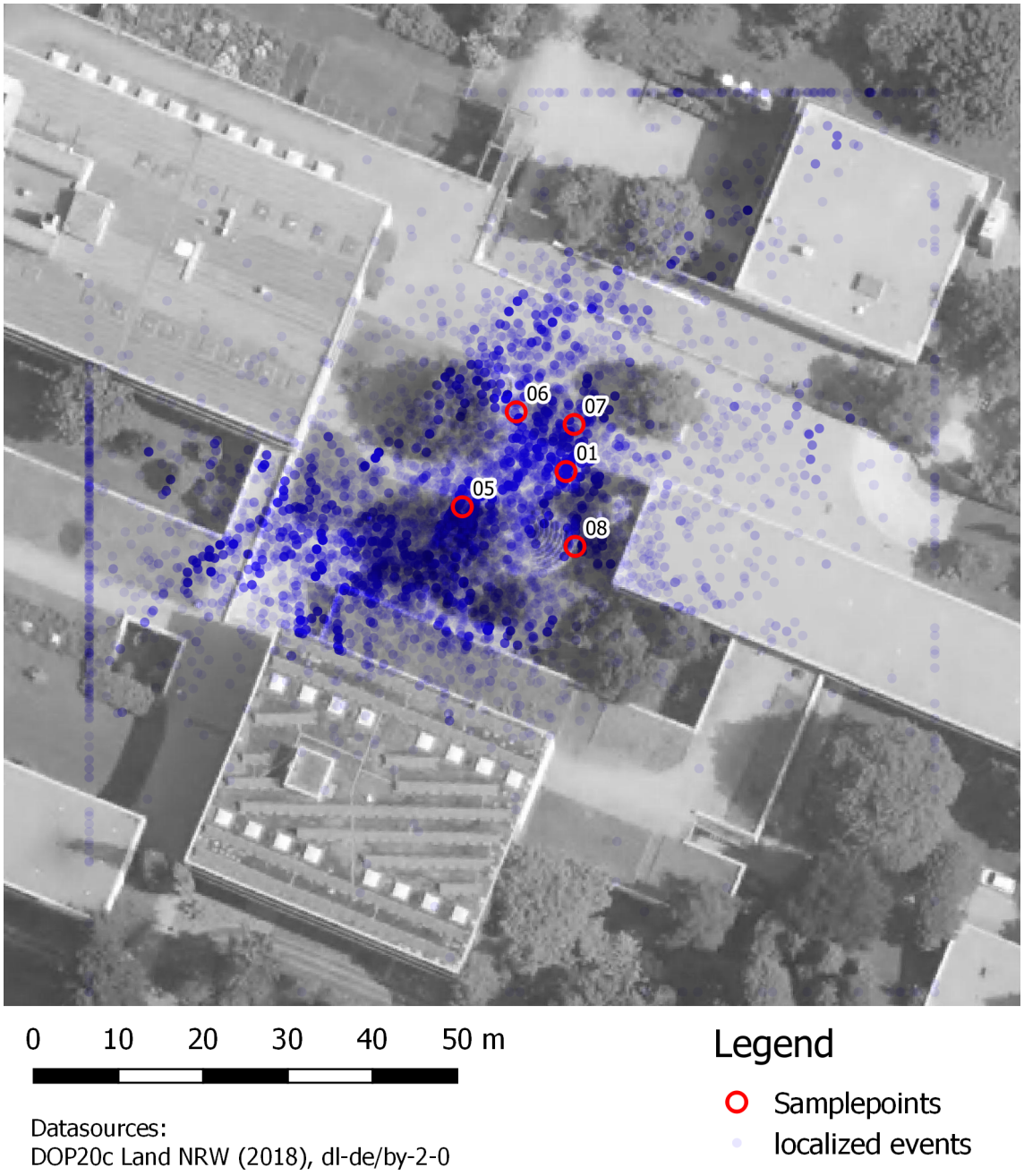

Abbildung 4.9: Karte der lokalisierten Events. Die lokalisierten Events sind als transparente blaue Marker über dem Orthofoto gezeichnet. Die roten Marker entsprechen den Abspielpositionen. Die Events sind 3D-Punkte, hier als 2D Darstellung.

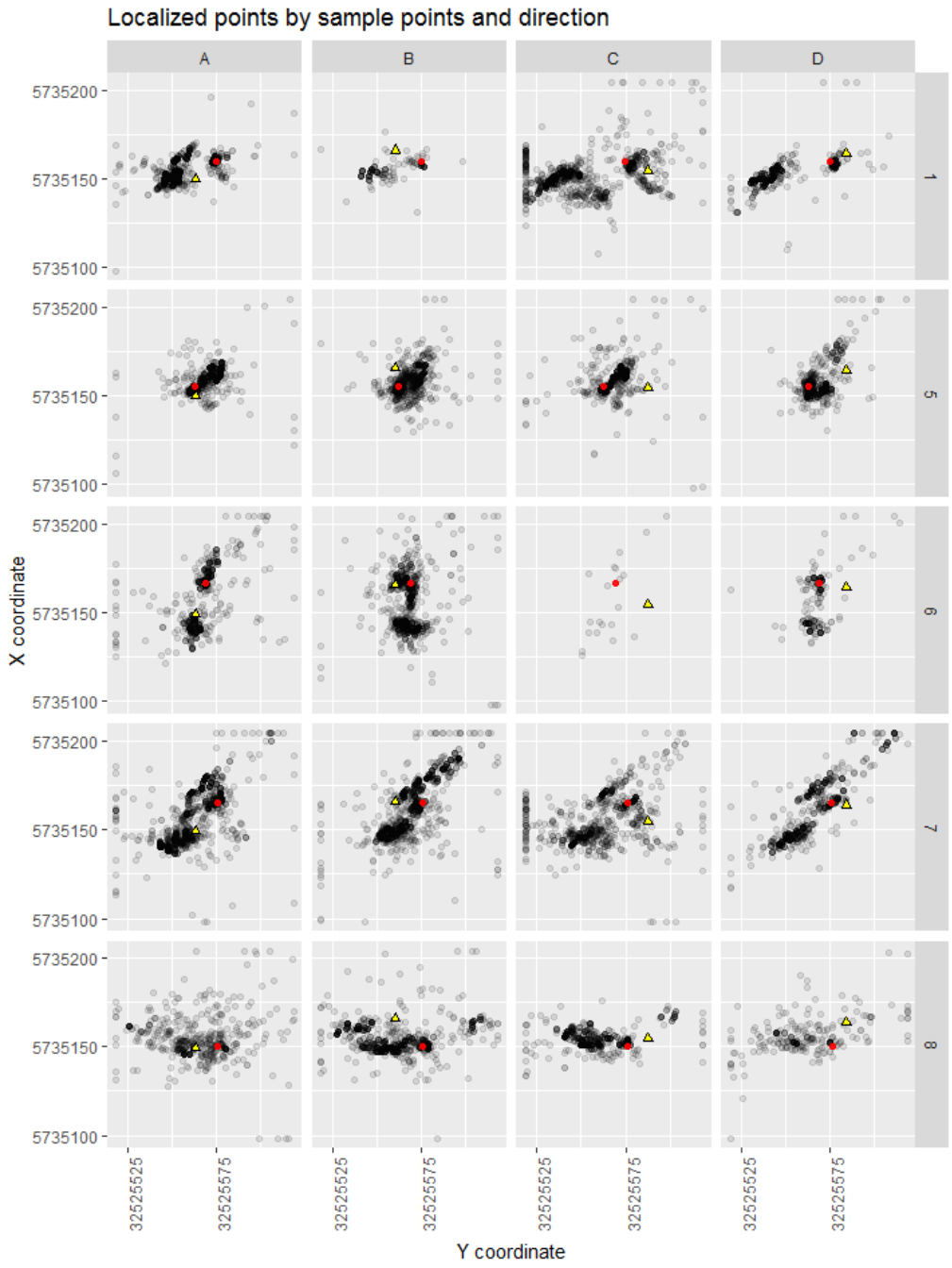

Abbildung 4.10: Karten der lokalisierten Events. Die lokalisierten Events sind als transparente schwarze Marker gezeichnet. Die roten Marker entsprechen den Abspielpositionen und die gelben Dreiecke entsprechen den mit dem Lautsprecher angepeilten Richtungsmarkern.

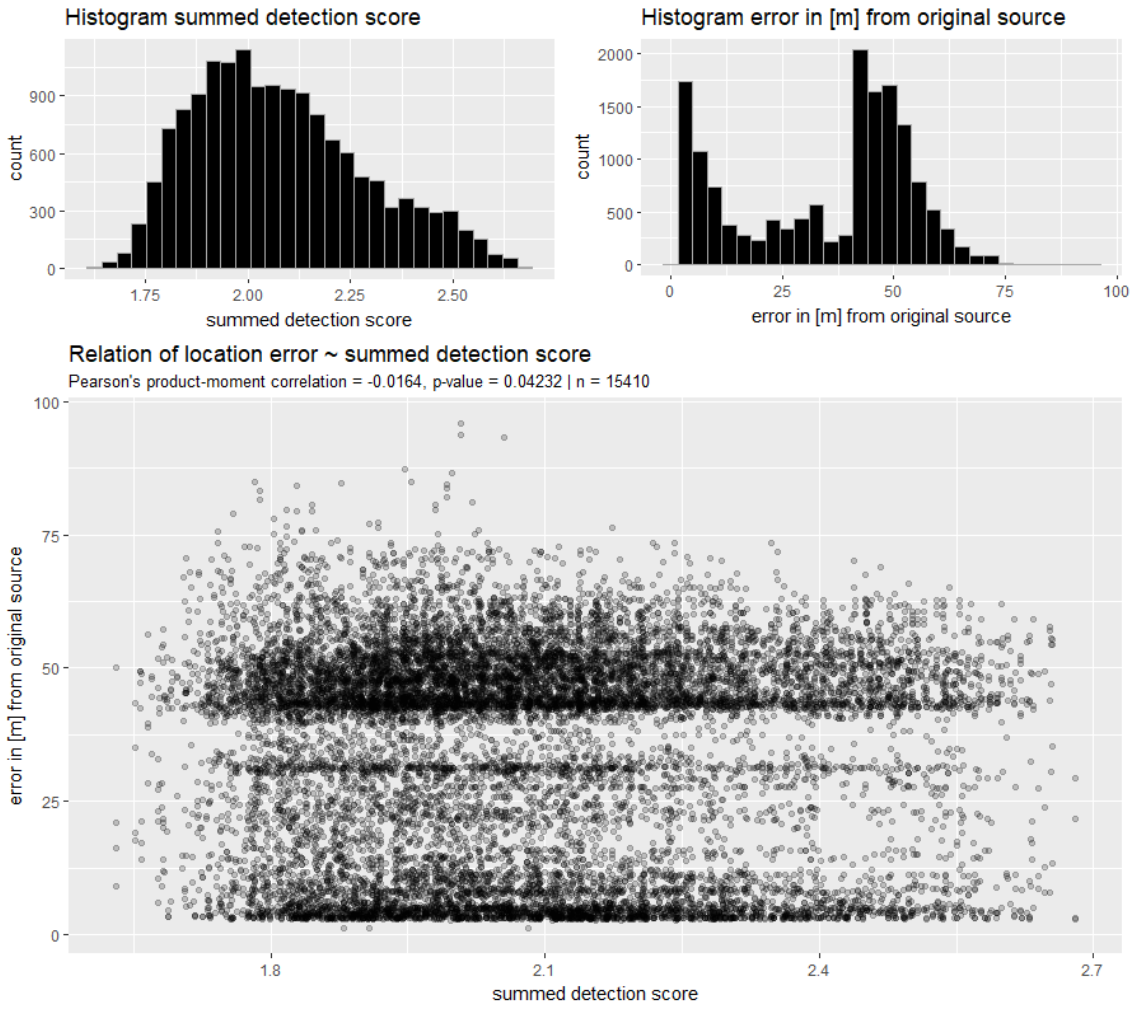

Abbildung 4.11: Zusammenhang zwischen Lokalisierungsgenauigkeit und Güte der Klassifikation. Es zeigt sich, mit einer Pearson's product-moment correlation von ∼-0.016 (p-value 0.042), eine signifikante, sehr schwache negative Korellation zwischen den untersuchten Größen.

von 0.05 abgelehnt werden muss. Jedoch kann für das 95% Konfidenzintervall angegeben werden, dass:

$$-3.21e^{-2} > cor > -5.67e^{-4} \tag{4.1}$$

und somit eine statistisch sehr schwache negative Korellation zwischen der Güte der Klassifikation und der Güte der Lokalisation vorliegt.

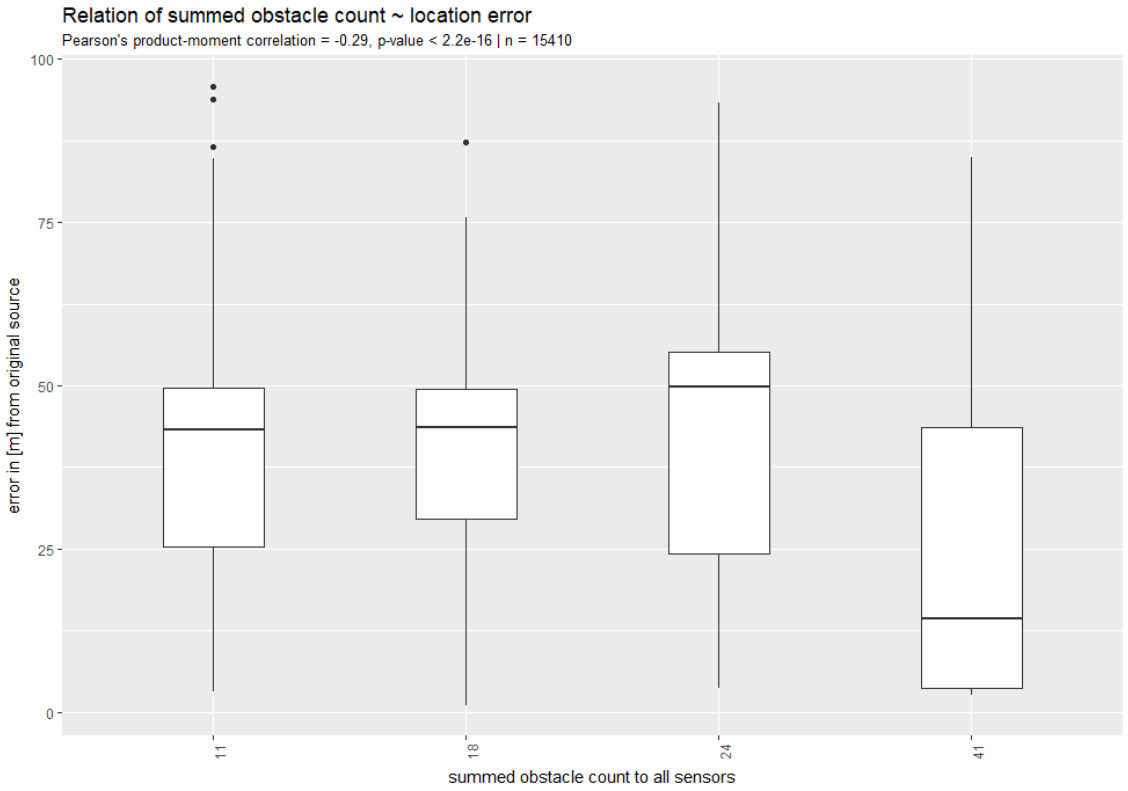

Abbildung 4.12: Zusammenhang zwischen Lokalisierungsgenauigkeit und dem Maß der Schallverschattung. Es zeigt sich, mit einer Pearson's product-moment correlation von -0.29 (p-value $< -2.2^{-16}$), eine signifikante negative Korellation zwischen den untersuchten Größen.

4.2.2 Einfluss der Schallverschattung

Die Hypothese *Die Güte der Lokalisation ist abhängig von der Schallverschattung des Schallgebers zum Mikrofon* kann bestätigt werden, jedoch nicht im erwarteten postiven Ausmaß. In Abbildung 4.12 zeigt sich, mit einer Pearson's product-moment correlation von ∼-0.29 und bei einem p-value von $< -2.2^{-16}$, dass die Nullhypothese H_0, *es liegt keine Korrelation vor*, bei einem α-Fehler von 0.05 abgelehnt werden muss. Für das 95% Konfidenzintervall kann angegeben werden, dass:

$$-0.30 > cor > -0.27 \tag{4.2}$$

und somit eine signifikante negative Korellation zwischen der Güte der Lokalisation und der Anzahl an schallverschattenden Objekten vorliegt.

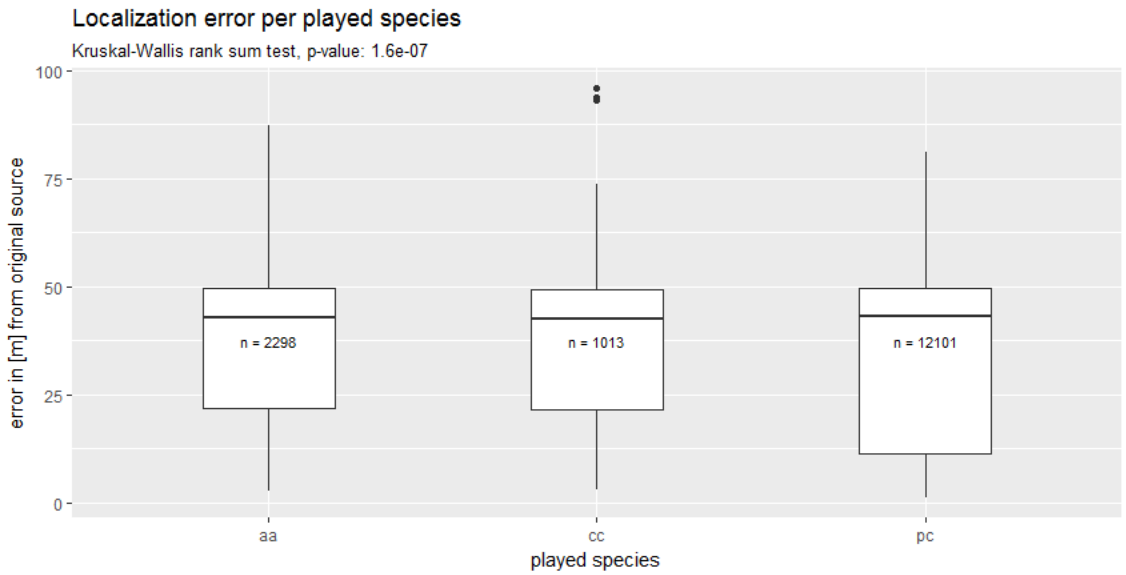

Abbildung 4.13: Zusammenhang zwischen Lokalisierungsgenauigkeit und der abgespielten Vogelart. Es zeigt sich bei einem Kruskal-Wallis rank sum test eine signifikante Abweichung zwischen den untersuchten Gruppen (p-Value 1.6^{-7}).

4.2.3 Einfluss der abgespielten Vogelart

Die Hypothese *Die Güte der Lokalisation ist abhängig von der Art des Schalls (Vogelart)* kann angenommen werden. Da die Lokalisierungsfehlerwerte zwischen den abgespielten Vogelarten keiner Normalverteilung folgen, wurde anstelle einer ANOVA ein Kruskal-Wallis rank sum test durchgeführt. Die Alternativhypothese H_0, *der Lokalisierungsfehler ist gleichverteilt zwischen den Gruppen*, kann mit einem p-Value von 1.6^{-7}, abgelehnt werden (siehe Abbildung 4.13).

4.2.4 Einfluss der Richtung des Schalls

Die Hypothese *Die Güte der Lokalisation ist abhängig von der Richtung des Schalls* kann abgelehnt werden. Die Nullhypothese H_0, *es liegt keine Korellation vor*, muss bei einem α-Fehler von 0.05 abgelehnt werden. Jedoch kann für das 95% Konfidenzintervall eine Pearson's product-moment correlation (cor):

$$-0.09 > cor > -0.05 \tag{4.3}$$

angegeben werden, wie in Abbildung 4.14 auf der nächsten Seite dargestellt. Hiermit liegt eine nicht signifikante negative Korellation vor.

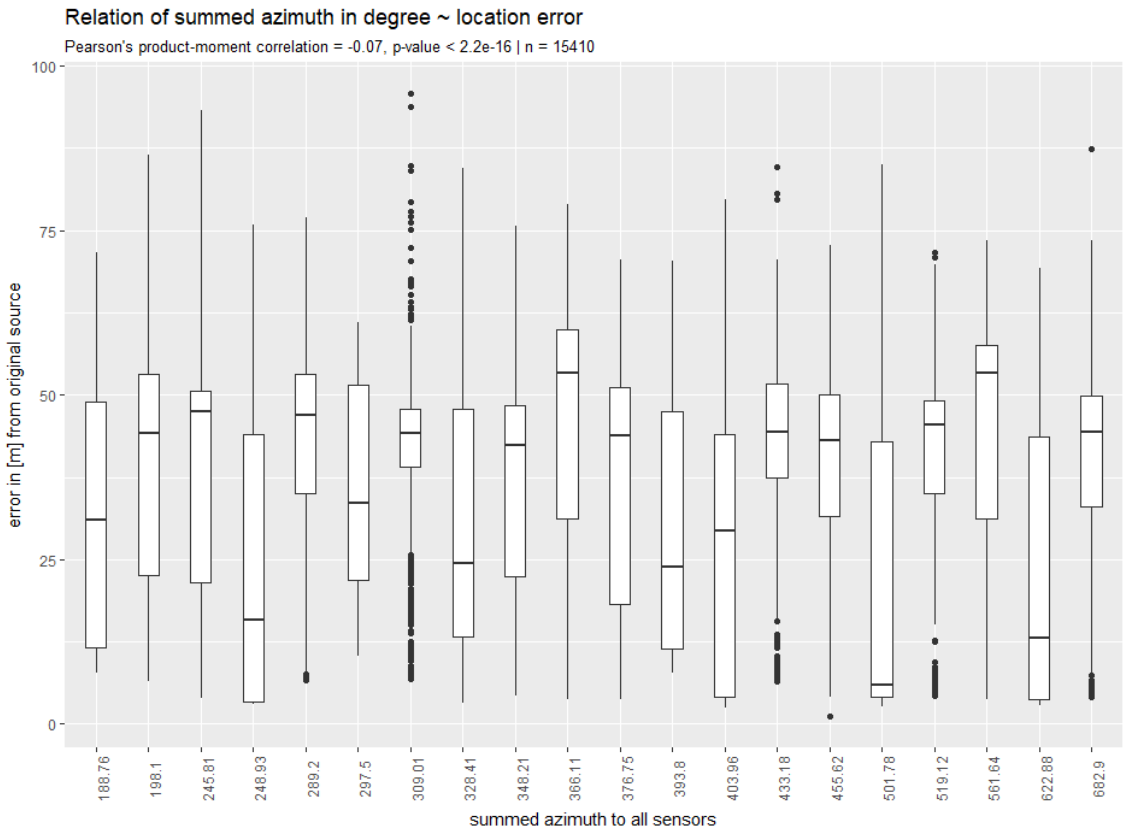

Abbildung 4.14: Zusammenhang zwischen Lokalisierungsgenauigkeit und dem summierten Abspielwinkel. Bei einer Pearson's product-moment correlation von -0.07 (p-value $< -2.2^{-16}$) liegt keine signifikante Korellation zwischen den untersuchten Größen vor.

5 | Diskussion

Das Kapitel Diskussion hinterfragt die verwendeten Methoden und Ergebnisse kritisch.

5.1 Methodenkritik

5.1.1 Aufnahmezeitpunkt

Die Aufnahme fand am 25.10.2017 und damit nicht unter gleichen phänologischen Bedingunen wie der in der Arbeit verwendete Laserscan statt. Die im Laserscan vorhandenen Rückgaben im voll beblätterten Zustand decken sich entsprechend nicht mit dem zum Untersuchungszeitpunkt vorgefundenen Zustand, wie in Abbildung 3.5 auf Seite 15 sichtbar. Dies beeinträchtigt die Aussagekraft der verwendeten Metrik für das Maß der Schallverschattung, wie in Abschnitt 3.5 auf Seite 23 beschrieben, negativ.

5.1.2 Fehlerhafte Daten

Die Aufnahmelogik des verwendeten PAMS sollte OS agnostisch erstellt werden. Daher wurde eine Umsetzung über *Python* und dem Modul *pyaudio* angestrebt. Diese wurde jedoch im Verlauf der Arbeit verworfen, da sich im Rahmen dieser unlösbare Probleme zeigten (Siehe IO ERROR Verweis in Zeile 7 des Listing A.3). Der Wechsel auf das *alsaaudio* machte Aufnahmen möglich, die jedoch intermittierende Fehlaufnahmen produzierten, wie in Abbildung 5.1 auf der nächsten Seite und Abbildung 5.2 auf der nächsten Seite dargestellt. Es konnte keine Regelmäßigkeit

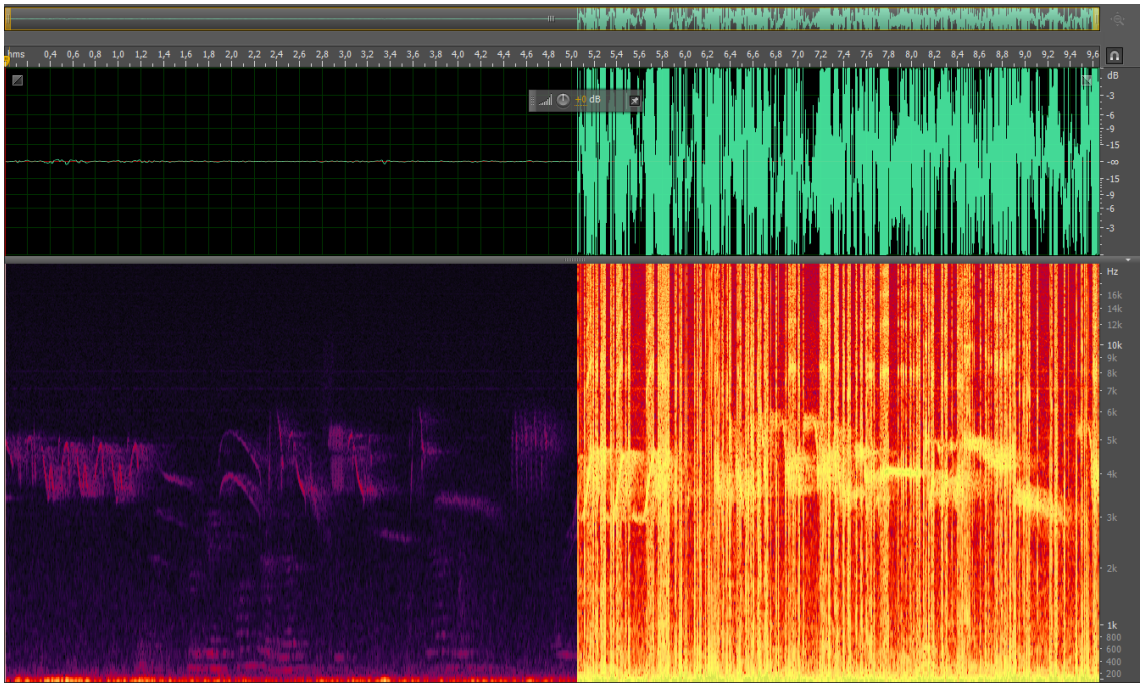

Abbildung 5.1: Spektrogramm und Wavefront Ansicht eines fehlerhaften Datensatzes.

zwischen den fehlerhaften Aufnahmen erkannt werden.

Da die Lokalisierungsalgorithmik Daten von allen vier Sensoren benötigt, macht ein Fehler eines Sensors die Aufnahme aller Sensoren dieses Zeitfensters unbrauchbar. Dies führte zum verwerfen von $\sim \frac{1}{5}$ der Aufnahmen, wie in Abbildung 5.3 auf der nächsten Seite gezeigt.

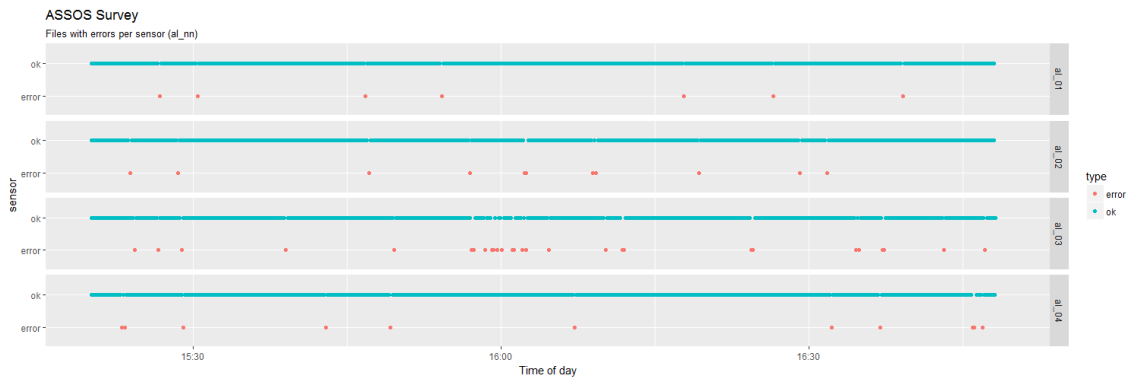

Abbildung 5.2: Fehlerhafte Dateien je Sensor.

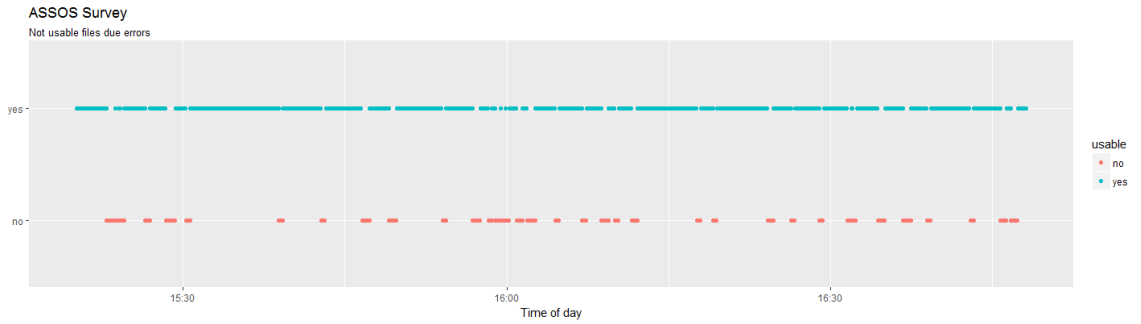

Abbildung 5.3: Gesamtheit der fehlerhaften Daten.

Tabelle 5.1: Offset bei der NTP Synchronisation zwischen Sensor und Hochschulneztwerk (ntp.hs-owl.de) abgefragt über den Unix Shell command *ntpq -p*.

remote	refid	st	t	when	poll	reach	delay	offset	jitter
*ntp-2.hs-owl.de	192.53.103.104	2	u	97	256	377	0.490	0.513	0.523
+ntp-1.hs-owl.de	130.149.17.8	2	u	29	256	377	0.518	1.375	0.442

5.1.3 Zeitsynchronisation

Die Synchronisation der *Raspberry Pi* wurde über NTP durchgeführt und zeigte eine maximales Offset von 1.375 ms je Sensor (siehe Tabelle 5.1). Im *worst case scenario* sind zwei Sensoren komplett gegenläufig asynchron, was zu einem maximalen Offset von 2.75 ms führt. Bei einer gemittelten Temperatur $\bar{T}$ von $\sim 14.8\,°C$ entspricht dies, über die Gleichung (3.2) auf Seite 27 berechnet, einer maximalen Abweichung von $\sim 0.93\,m$.

5.1.4 Lokalisierungsalgorithmus

Die in den erfassten Daten hohen Lokalisierungsfehler (siehe Abschnitt 4.1.2 auf Seite 32) übersteigen die in Wilson et al. (2014) $\bar{x} = 4.3$ m (2D) und Stepanian et al. (2016) $\bar{x} < 10$ m (3D) genannten Werte um ein Vielfaches ($\bar{x} = 34.96$ m, 3D). Die hohen Fehlerwerte sind trotz Tachymetereinmaß der Sensor- und Signalposition und damit einhergehender höheren Lagegenauigkeit (cm) der dort verwendeten GPS gestützten Verfahren zur Sensor- und Signalverortung (Wilson et al. (2014) max 2.51 2D; Stepanian et al. (2016) $\pm 8.66\,m$ 3D) zu beobachten. Welche Faktoren diese niedrige Güte der Lokalisierungs bedingen konnte nicht abschließend geklärt werden.

Der in WILSON ET AL. (2014) genutzte Algorithmus wurde in seiner *R* Variante getestet und verworfen, da keine Verortung durchgeführt werden konnte. Die localize Methode konnte auf Grund eines Matrix Berechnungsfehlers nicht durchgeführt werden. Der daher in der Untersuchung genutzte Algorithmus zeigt eine Heteroskedastizität der Relation Standardabweichung (σ) zum verbleibenden Fehler (Abbildung 2.5 auf Seite 11), welche zu einer abnehmenden Korellation zwischen den Faktoren bei steigendem σ führt. Durch Messfehler der Sensoren kann somit für den Suchpunkt (SP) kein optimaler Standort ($SP_{[x,y]fit} = 0$) definiert werden. Entsprechend liegt der *best fit* Suchpunkt mit einer höheren verbleibenden σ und entsprechend schlechterer Lokalisierungsgenauigkeit vor.

Die in Abbildung 4.6 auf Seite 35 multimodale Verteilungskurve könnte durch die im Lokalisierungsalgorithmus verwendeten Bounds (Listing A.8, Zeile 313) entstanden sein. Die Bounds der Fit-Funktion führen zu einer Häufung von Lokalisierungen an den Grenzen des Untersuchungsbereiches, wie in Abbildung 4.9 auf Seite 38 gezeigt.

5.1.5 Sensor Platzierung

Im Rahmen der Validierung des Lokalisierungsalgorithmus (siehe Abschnitt 2.3 auf Seite 7) sind verschiedene Sensor Platzierungen simuliert worden. Hierbei hat sich gezeigt, dass die Platzierung der Sensoren massiven Einfluss auf die Lokalisierungsgenauigkeit besitzt, wie in Abbildung 5.4 auf der nächsten Seite beispielhaft gezeigt. Bei dem schiefe Ebene Aufbau werden die Lokalisierten Punkte nur auf dieser Ebene verortet. Bei dem irregulären Aufbau können die Signale mit hoher Präzision verortet werden.

5.2 Ergebnisse

Die genannten kritischen Punkte der verwendeten Methoden stellen die erstellten Analysen und Auswertungen in Frage. Die in Abbildung 4.11 auf Seite 40 klar sichtbaren Datenpunktcluster lassen vermuten, dass bisher nicht untersuchte Faktoren die Lokalisierungsgenauigkeit beeinflusst haben. Die insgesamt niedrige Lokalisierungsgüte und die überwiegend nicht normal verteilten Datensätze verstärken diese Vermutung.

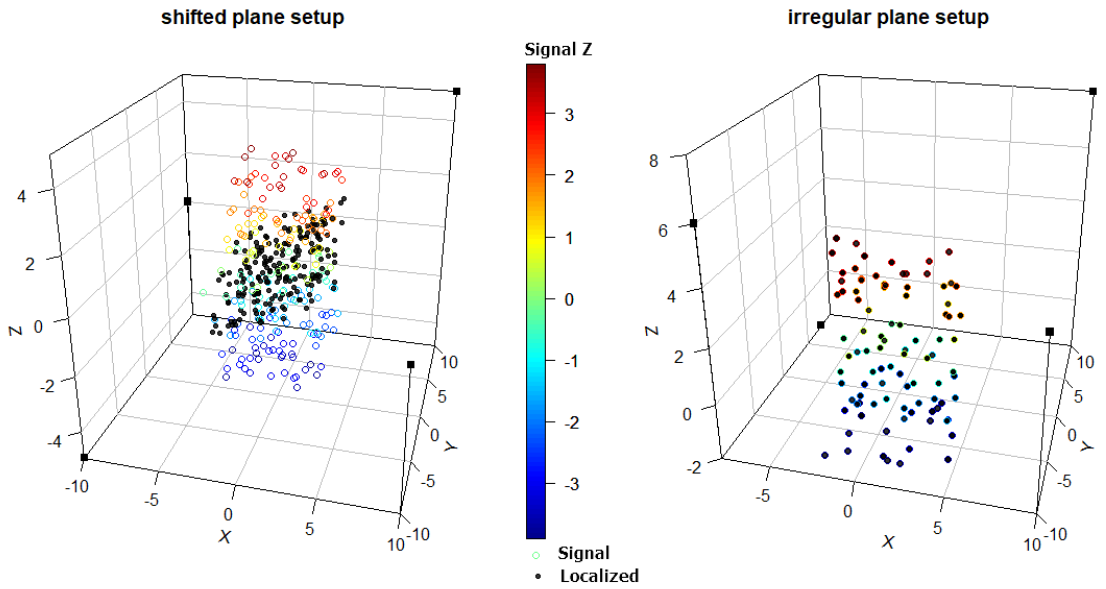

Abbildung 5.4: Verortungsgenauigkeit von zwei simulierten Sensor Platzierungen - links die Sensor Platzierung in einer schiefen Ebene und rechts als irreguläre Oberfläche (siehe Tabelle 5.2). Die schwarzen Quader stellen die Sensoren, die farbigen Punkte die Signale und die schwarzen Punkte die Lokalisationen dar.

Tabelle 5.2: Koordinaten der Sensor Platzierungen der in Abbildung 5.4 gezeigten Simulation

Typ	Sensor	x	y	z
schiefe Ebene	a	10	10	5
	b	-10	10	0
	c	10	-10	0
	d	-10	-10	-5
irreguläre Ebene	a	10	10	8
	b	-8	8	-2
	c	10	-10	4
	d	-8	-10	6

Das die Güte der Klassifikation keinen maßgeblichen Einfluß auf die Lokalisierungsgüte besitzt (siehe Abschnitt 4.2.1) kann nachvollzogen werden. Der verwendete Lokalisierungsalgorithmus verwendet lediglich die TDOA. Hierbei ist nicht relevant, wie passend der klassifizierte Audiobereich mit dem gesuchtem Audiosignal ist.

Die in Abschnitt 4.2.2 aufgezeigte negative Korellation zwischen der Anzahl an schallverschattenden Objekten und der Lokalisierungsgenauigkeit scheint keiner Kausalität zu folgen. Die genutzten Aufnahmepositionen haben letztlich eine Population von nur vier verschiedene Datenpunkten an Schallverschattung ergeben (11, 18, 24, 41) und die verwendete Metrik für das Maß der Schallverschattung muss kritisch hinterfragt werden, wie in Abschnitt 5.1.1 erläutert.

Der Einfluß der abgespielten Vogelart (siehe Abschnitt 4.2.3 auf Seite 42) zeigt, dass *Phylloscopus collybita* (pc) signifikant besser zu verorten ist. Die im Vergleich zu den anderen Arten sehr einfache und kurze Strophe scheint bei der Erstellung der Audio-Events von Vorteil zu sein. Hierbei ist jedoch einschränkend anzumerken, dass 17 % der pc Samples false-positives waren.

Wie in Abschnitt 4.2.4 auf Seite 42 analysiert, hat die Kopfausrichtung des Vogels keinen signifikanten Einfluß auf die Verortungsgenauigkeit. Kritisch zu betrachten ist hierbei jedoch, dass alle Abspielpositionen innerhalb des Sensornetzwerkes lagen. Gegebenfalls ist die Kopfausrichtung (und damit die Schallrichtung) bei Audio-Events außerhalb des Netzwerkes von Bedeutung.

6 | Ausblick

Im Kapitel Ausblick werden Anpassungmöglichkeiten der verwendeten Methodik aufgezeigt und weiterer Forschungsbedarf genannt.

6.1 Aufnahmelogik

Die für die Datenerhebung verwendete Aufnahmelogik (siehe Abschnitt 3.2 auf Seite 14) hat nicht brauchbare Sounddateien produziert. Weitere Tests haben gezeigt, dass eine über die ALSA CLI aufgerufene Aufnahme fehlerfrei aufnimmt. Ensptrechend liegt nahe den in dieser Arbeit verwendeten Aufnahmecode kritisch zu hinterfragen. Das in WHYTOCK & CHRISTIE (2017) verwendete System nutzt ebenfalls die ALSA CLI und wurde bereits erfolgreich angewandt.

6.2 Klassifikatoren

Die verwendeten Standartwerte für das Hanning Window und dessen Overlap können weitergehend angepasst werden. WILSON ET AL. (2014) nutzt beispielsweise ein Overlap von 87,5 % , anstelle der in dieser Arbeit verwendeten 0 %, mit guten Ergebnissen bei der Klassifikation. Die Güte der erstellten Templates kann iterativ über einen vorhandenen Testdatensatz evaluiert und die verwendeten Parameter (Window Typ, length, Overlap; score cut-off, frequency limits, Rectangle Selection) entsprechend modifiziert werden.

Neben dem verwendeten Spectorgramm Cross-Corellation Klassifikator können andere Klassifikatoren, wie zum Beispiel binary point matching (KATZ ET AL.,

2016b), oder feature-based Klassifikatoren (GIANNAKOPOULOS, 2015), am vorhandenen Testdatensatz evaluiert werden, um die Klassifikationserfolge zu optimieren.

6.3 Aufbau

Wie in Abschnitt 5.1.5 auf Seite 47 diskutiert, scheint die Positionierung der Sensoren des Netzwerkes maßgbelichen Einfluß auf die Lokalisierungsgenauigkeit zu besitzen. Die erarbeitete Simulationsumgebung sollte genutzt werden, um diese Vermutung weitergehend zu untersuchen und zukünftige Sensornetzwerke entsprechend optimiert zu planen.

Des Weiteren kann über die Simulationsumgebung der Einfluss der Anzahl von Sensoren auf die Lokalisierungsgüte unter unterschiedlich starken Signal-To-Noise Umgebungen untersucht werden.

Neben der Positionierung der Sensoren sollte auch die Signalpositionierung erweitert untersucht werden, so zum Beispiel Signalpositionen außerhalb des Netzwerkes.

6.4 Lokalisierunsalgorithmus

Nebem dem in dieser Arbeit verwendeten *Limited-memory BFGS* (siehe Abschnitt 3.6 auf Seite 24) Minimierungsalgorithmus sollten weitere Verfahren des *Scipy.Minimize* Moduls auf ihre Performance und Lokalisierungsgüte untersucht werden.

Wie in Abschnitt 2.3 auf Seite 7 aufgezeigt, ist eine Vielzahl weiterer Verfahren zur Lokalisation möglich. Dementsprechend sollte eine kritische Zusammenschau verschiedener Algorithmen, unter gleichen Bedingungen, stattfinden.

Die Scatterplots in Abbildung 4.10 zeigen nicht identisches Streuungsverhalten, unterteilt nach Abspielrichtung und -ort. Welche Faktoren Einfluß auf die Streuung nehmen ist bisher noch unbekannt und sollte untersucht werden.

6.5 Component-Based-Architecture

Da sowohl die Software Komponenten, als auch die Hardware Komponenten eines PAMS divers sind, sollten künftige Entwicklungen angelehnt an eine Component-Based-Architecture (CBA) in seiner Hardware- und Softwarearchitektur modular entwickelt werden. Datenbanken wären demzufolge über Object-Relational-Mapping (ORM) Funktionen angesprochen und damit austauschbar sowie weite Teile der Software über zum Beispiel Docker Images, als Anwendungscontainer, realisiert. Hierdurch könnten die derzeit raschen und hochdynamischen Entwicklungen der Klassifizierungs- und Lokalisierungsalgorithmen in bestehenden PAMS gegen ihre jeweils verbesserten Varianten ausgetauscht werden.

6.6 Sensor Observation Service

Neben dem Verwenden einer CBA sollte zur Sicherung eines interoperablen Datenaustausches die Implementierung der Sensor-Observation-Service OGC-Spezifikation angestrebt werden, wie in CANNATA ET AL. (2015); CHEN ET AL. (2009) gezeigt. Hierbei gilt es für bioakustische Datensätze passende *Observation & Measurement* Schemata (INSPIRE MAINTENANCE AND IMPLEMENTATION GROUP (MIG), 2016) zu definieren und umzusetzen.

Literaturverzeichnis

ACEVEDO, M. A., CORRADA-BRAVO, C. J., CORRADA-BRAVO, H., VILLANUEVA-RIVERA, L. J. & AIDE, T. M. (2009): *Automated classification of bird and amphibian calls using machine learning: A comparison of methods*. In: Ecological Informatics, **4**, 4: 206–214. URL http://linkinghub.elsevier.com/retrieve/pii/S1574954109000351.

AIDE, T. M., CORRADA-BRAVO, C., CAMPOS-CERQUEIRA, M., MILAN, C., VEGA, G. & ALVAREZ, R. (2013): *Real-time bioacoustics monitoring and automated species identification*. In: PeerJ, **1**: e103. URL https://peerj.com/articles/103.

ALI, A. M., ASGARI, S., COLLIER, T. C., ALLEN, M., GIROD, L., HUDSON, R. E., YAO, K., TAYLOR, C. E. & BLUMSTEIN, D. T. (2009): *An empirical study of collaborative acoustic source localization*. In: Journal of Signal Processing Systems, **57**, 3: 415–436. URL https://www.eeb.ucla.edu/Faculty/Blumstein/pdfreprints/Ali_etal_2009_JSPS.pdf.

BMUB (2017): *Ressortforschungsplan 2018*. Techn. Ber., Bundesministeriums für Umwelt, Naturschutz, Bau und Reaktorsicherheit (BMUB), Berlin. URL http://www.bmub.bund.de/fileadmin/Daten_BMU/Download_PDF/Forschung/ressortforschungsplan_gesamt_2018_bf.pdf.

BROWNING, E., GIBB, R., GLOVER-KAPFER, P. & JONES, K. (2017): *Passive acoustic monitoring in ecology and conservation*. URL https://www.researchgate.net/publication/320323376_Passive_acoustic_monitoring_in_ecology_and_conservation.

BUAKA MUANKE, P. & NIEZRECKI, C. (2007): *Manatee position estimation by passive acoustic localization*. In: The Journal of the Acoustical Society of America, **121**, 4: 2049–2059. URL http://asa.scitation.org/doi/10.1121/1.2532210.

BUNDESREPUBLIK DEUTSCHLAND (2009): *Gesetz über Naturschutz und Landschaftspflege (Bundesnaturschutzgesetz - BNatSchG); §6 Beobachtung von Natur und Landschaft*. URL http://www.gesetze-im-internet.de/bnatschg_2009/__6.html.

CANNATA, M., ANTONOVIC, M., MOLINARI, M. & POZZONI, M. (2015): *istSOS, a new sensor observation management system: software architecture and a real-case application for flood protection*. In: Geomatics, Natural Hazards and Risk, **6**, 8: 635–650. URL http://www.tandfonline.com/doi/full/10.1080/19475705.2013.862572.

CHEN, N., DI, L., YU, G. & MIN, M. (2009): *A flexible geospatial sensor observation service for diverse sensor data based on Web service*. In: ISPRS Journal of Photogrammetry and Remote Sensing, **64**, 2: 234–242. URL http://linkinghub.elsevier.com/retrieve/pii/S0924271608001160.

DIGBY, A., TOWSEY, M., BELL, B. D. & TEAL, P. D. (2013): *A practical comparison of manual and autonomous methods for acoustic monitoring*. In: Methods in Ecology and Evolution, **4**, 7: 675–683. URL http://doi.wiley.com/10.1111/2041-210X.12060.

EUROPEAN COMMISSION - GENERAL DIRECTION ENVIRONMENT (2007): *Guidance document on the strict protection of animal species of Community interest under the Habitats Directive 92/43/EEC*. URL http://ec.europa.eu/environment/nature/conservation/species/guidance/pdf/guidance_en.pdf.

FROMMOLT, K. H. & TAUCHERT, K. H. (2014): *Applying bioacoustic methods for long-term monitoring of a nocturnal wetland bird*. In: Ecological Informatics, **21**: 4–12. URL http://www.sciencedirect.com/science/article/pii/S1574954113001301.

GIANNAKOPOULOS, T. (2015): *PyAudioAnalysis: An open-source python library for audio signal analysis*. In: PLoS ONE, **10**, 12: e0144610. URL http://dx.plos.org/10.1371/journal.pone.0144610.

GOËAU, H., GLOTIN, H., VELLINGA, W.-P., PLANQUÉ, R. & JOLY, A. (2016): *LifeCLEF Bird Identification Task 2016: The arrival of Deep learning*. In: *Working Notes of CLEF 2016 - Conference and Labs of the Evaluation forum*, 440–449. URL https://hal.archives-ouvertes.fr/hal-01373779/document.

HEDLEY, R. W., HUANG, Y. & YAO, K. (2017): *Direction-of-arrival estimation of animal vocalizations for monitoring animal behavior and improving estimates of abundance*. In: Avian Conservation and Ecology, **12**, 1: art6. URL http://www.ace-eco.org/vol12/iss1/art6/.

HILL, A. P., PRINCE, P., COVARRUBIAS, E. P., DONCASTER, C. P., SNADDON, J. L. & ROGERS, A. (2017): *AudioMoth: Evaluation of a smart open acoustic device for monitoring biodiversity and the environment*. In: Methods in Ecology and Evolution.

HOEDT, F. (2017): *ASSOS Listen Github Repository*. URL https://doi.org/10.5281/zenodo.1200264.

HOEDT, F. (2018a): *ASSOS Classifier Datasets*. URL https://doi.org/10.5281/zenodo.1200270.

HOEDT, F. (2018b): *ASSOS Geodata*. URL https://doi.org/10.5281/zenodo.1200358.

HOEDT, F. (2018c): *ASSOS recorded data*. URL https://doi.org/10.5281/zenodo.1200464.

HORNSTEIN, J., LOPES, M., SANTOS-VICTOR, J. & LACERDA, F. (2006): *Sound Localization for Humanoid Robots - Building Audio-Motor Maps based on the HRTF*. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1170–1176. IEEE. URL http://ieeexplore.ieee.org/document/4058525/.

IMMISCH, L. & WILSTRUP, C. (2017): *pyalsaaudio 0.8.4*. URL https://pypi.python.org/pypi/pyalsaaudio.

IMRAN, M., HUSSAIN, A., QAZI, N. M. & SADIQ, M. (2016): *A methodology for sound source localization and tracking: Development of 3D microphone array for near-field and far-field applications*. In: *2016 13th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 586–591. IEEE. URL http://ieeexplore.ieee.org/document/7429936/.

INSPIRE MAINTENANCE AND IMPLEMENTATION GROUP (MIG) (2016): *Guidelines for the use of Observations & Measurements and Sensor Web Enablement-related standards in INSPIRE | INSPIRE*. URL http://inspire.ec.europa.eu/id/document/tg/d2.9-o%26m-swe.

KATZ, J., HAFNER, S. D. & DONOVAN, T. (2016a): *Assessment of Error Rates in Acoustic Monitoring with the R package monitoR*. In: Bioacoustics, **25**, 2: 177–196. URL http://www.tandfonline.com/doi/full/10.1080/09524622.2015.1133320.

KATZ, J., HAFNER, S. D. & DONOVAN, T. (2016b): *Tools for automated acoustic monitoring within the R package monitoR*. In: Bioacoustics, **25**, 2: 197–210. URL http://www.tandfonline.com/doi/full/10.1080/09524622.2016.1138415.

KOGAN, J. A. & MARGOLIASH, D. (1998): *Automated recognition of bird song elements from continuous recordings using dynamic time warping and hidden Markov models : A comparative study*. In: The Journal of the Acoustical Society of America, **103**, 4: 2185–2196. URL http://asa.scitation.org/doi/10.1121/1.421364.

LAMBERT, K. T. A. & MCDONALD, P. G. (2014): *A low-cost, yet simple and highly repeatable system for acoustically surveying cryptic species*. In: Austral Ecology, **39**, 7: 779–785. URL http://onlinelibrary.wiley.com/doi/10.1111/aec.12143/abstract.

LAND NRW (2017): *Digitales Oberflächenmodell mittlerer Punktabstand 1m*. URL https://www.opengeodata.nrw.de/produkte/geobasis/dom/dom1l/.

LASSECK, M. (2015): *Towards Automatic Large-Scale Identification of Birds in Audio Recordings*. In: *LNCS*, Bd. 9283, 364–375. URL http://link.springer.com/10.1007/978-3-319-24027-5_39.

LI, X., DANIEL DENG, Z., RAUCHENSTEIN, L. T. & CARLSON, T. J. (2016): *Contributed Review: Source-localization algorithms and applications using time of arrival and time difference of arrival measurements*. In: Citation: Review of Scientific Instruments, **87**. URL https://aip.scitation.org/doi/10.1063/1.4947001.

MENNILL, D. J., BATTISTON, M., WILSON, D. R., FOOTE, J. R. & DOUCET, S. M. (2012): *Field test of an affordable, portable, wireless microphone array for spatial monitoring of animal ecology and behaviour*. In: Methods in Ecology and Evolution, **3**, 4: 704–712. URL http://doi.wiley.com/10.1111/j.2041-210X.2012.00209.x.

MENNILL, D. J., BURT, J. M., FRISTRUP, K. M. & VEHRENCAMP, S. L. (2006): *Accuracy of an acoustic location system for monitoring the position of duetting songbirds in tropical forest*. In: The Journal of the Acoustical Society of America, **119**, 5: 2832–2839. URL http://asa.scitation.org/doi/10.1121/1.2184988.

MILLS, D. (1991): *Internet time synchronization: the network time protocol*. In: IEEE Transactions on Communications, **39**, 10: 1482–1493. URL http://ieeexplore.ieee.org/document/103043/.

MILLS, H. (2000): *Geographically distributed acoustical monitoring of migrating birds*. In: The Journal of the Acoustical Society of America, **108**, 5: 2582–2582. URL http://asa.scitation.org/doi/10.1121/1.4743594.

MUSEUM FÜR NATURKUNDE BERLIN (2017): *Tierstimmenarchiv des Museums für Naturkunde der Humboldt-Universität zu Berlin*. URL http://www.tierstimmenarchiv.de.

PTACEK, L., MACHLICA, L., LINHART, P., JASKA, P. & MULLER, L. (2016): *Automatic recognition of bird individuals on an open set using as-is recordings*. In: Bioacoustics, **25**, 1: 55–73. URL http://dx.doi.org/10.1080/09524622.2015.1089524.

R CORE TEAM (2018): *R: The R Project for Statistical Computing*. URL https://www.r-project.org/.

R FOUNDATION (2018): *R: The R Project for Statistical Computing*. URL https://www.r-project.org/.

RANJARD, L., REED, B. S., LANDERS, T. J., RAYNER, M. J., FRIESEN, M. R., SAGAR, R. L. & DUNPHY, B. J. (2017): *MatlabHTK: a simple interface for bioacoustic analyses using hidden Markov models*. In: Methods in Ecology and Evolution, **8**, 5: 615–621. URL http://doi.wiley.com/10.1111/2041-210X.12688.

RASPBERRY.ORG (2018): *Raspberry Pi 3 Model B - Raspberry Pi*. URL https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

ROSS, J. C. & ALLEN, P. E. (2014): *Random Forest for improved analysis efficiency in passive acoustic monitoring*. In: Ecological Informatics, **21**: 34–39. URL https://www.sciencedirect.com/science/article/pii/S1574954113001234.

SHONFIELD, J. & BAYNE, E. M. (2017): *Autonomous recording units in avian ecological research: current use and future applications*. In: Avian Conservation and Ecology, **12**, 1: art14. URL http://www.ace-eco.org/vol12/iss1/art14/.

STEPANIAN, P. M., HORTON, K. G., HILLE, D. C., WAINWRIGHT, C. E., CHILSON, P. B. & KELLY, J. F. (2016): *Extending bioacoustic monitoring of birds aloft through flight call localization with a three-dimensional microphone array*. In: Ecology and Evolution, **6**, 19: 7039–7046.

THE SCIPY COMMUNITY (2017): *scipy.optimize.minimize — SciPy v0.19.0 Reference Guide*. URL https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.optimize.minimize.html.

VENTURA, T. M., DE OLIVEIRA, A. G., GANCHEV, T. D., DE FIGUEIREDO, J. M., JAHN, O., MARQUES, M. I. & SCHUCHMANN, K.-L. (2015): *Audio parameterization with robust frame selection for improved bird identification*. In: Expert Systems with Applications, **42**, 22: 8463–8471. URL http://linkinghub.elsevier.com/retrieve/pii/S0957417415004625.

WEBOFSCIENCE (2018): *Web of Science [v.5.27] - Result Analysis*. URL https://wcs.webofknowledge.com/RA/analyze.do.

WHYTOCK, R. C. & CHRISTIE, J. (2017): *Solo: an open source, customizable and inexpensive audio recorder for bioacoustic research*. URL http://dx.doi.org/10.1111/2041-210X.12678.

WILDLIFE ACOUSTICS (2018): *Wildlife Acoustics - About Wildlife Acoustics*. URL https://www.wildlifeacoustics.com/company.

WILSON, D. R., BATTISTON, M., BRZUSTOWSKI, J. & MENNILL, D. J. (2014): *Sound Finder: a new software approach for localizing animals recorded with a microphone array*. In: Bioacoustics, **23**, 2: 99–112. URL http://www.tandfonline.com/doi/abs/10.1080/09524622.2013.827588.

ZHAO, Z., ZHANG, S.-H., XU, Z.-Y., BELLISARIO, K., DAI, N.-H., OMRANI, H. & PIJANOWSKI, B. C. (2017): *Automated bird acoustic event detection and robust species classification*. In: Ecological Informatics, **39**: 99–108. URL http://www.sciencedirect.com/science/article/pii/S157495411630231X.

A | Anhang

A.1 WebOfScience Abfrage

Die in Abbildung 1.1 auf Seite 2 gezeigte Auswertung basiert auf folgender am 01.02.2018 ausgeführten Abfrage auf WEBOFSCIENCE (2018):

```
TOPIC: (acoustic recording)
OR TOPIC: (passive acoustic monitoring)
OR TOPIC: (bioacoustic monitoring)
OR TOPIC: (automated digital recording system)
OR TOPIC: (autonomous recording)
OR TOPIC: (autonomous recorder)
OR TOPIC: (autonomous recording unit)
AND DOCUMENT TYPES: (Article)
AND YEAR PUBLISHED: (2005-2018)

Analysis: WEB OF SCIENCE CATEGORIES:
(ACOUSTICS OR COMPUTER SCIENCE INFORMATION SYSTEMS
OR ENVIRONMENTAL SCIENCES OR REMOTE SENSING
OR BIODIVERSITY CONSERVATION)
AND WEB OF SCIENCE CATEGORIES: (ECOLOGY)
```

A.2 ASSOS LISTEN Image

Das in Abschnitt 3.1 auf Seite 12 genannte Debian Stretch Image der Sensoren wurde wie folgt aufgebaut:

Listing A.1: Berechnung der Distanz aus den Delta Zeitstempeln

```
###### Documentation for the ASSOS LISTEN Raspberry Pi 3A image
# burn stretch lite image
# put ssh file (emtpy, without extension) on sdcard to enable
    ↪ ssh in headless mode
# put wlan conf file onto sdcard for headless WLAN startup
sudo apt-get update
sudo apt-get upgrade -y

# get python headers, pip and git
sudo apt-get install python-dev python-pip git -y
# get headers for alsa to compile python alsa
sudo apt-get install libasound2-dev -y

# no chache dir since raspi does not have enough RAM for
    ↪ building
# user to get write permission to the dist package folder
pip --no-cache-dir install pyalsaaudio --user

# pyaudio libraries and prerequisites for scipy
sudo apt-get install python-pyaudio libblas-dev liblapack-dev
    ↪ libatlas-base-dev gfortran -y

# as prerequisites for pyAudioAnalysis
pip --no-cache-dir install numpy matplotlib scipy sklearn
    ↪ hmmlearn simplejson eyeD3 pydub pathlib --user

# clone into pyAudioAnalysis
pip --no-cache-dir install pyAudioAnalysis --user

# get ffmpeg for debian stretch
sudo apt-get install libav-tools -y

# get example files for pyAudioAnalysis
```

```
30 git clone https://github.com/tyiannak/pyAudioAnalysis.git
31
32 # get paura
33 sudo apt-get install python-alsaaudio python-opencv -y
34 git clone https://github.com/tyiannak/paura.git
35
36 # configure microphone
37 # change or create /etc/modprobe.d/alsa-base.conf
38 #   to use the usb-sounddevice as default
39 # https://raspberrypi.stackexchange.com/questions/40831/
40 #      how-do-i-configure-my-sound-for-jasper-on-raspbian-jessie
41 # change the sensitivity of the microphone:
42 alsamixer -V capture
```

A.3 Aufnahmelogik

Die in Abschnitt 3.1 auf Seite 12 angesprochene Aufnahmelogik wurde in *Python* Programmiert und wird als *Github Repository* gemanaged (siehe (HOEDT, 2017)). Der zum Aufnahmezeitpunkt verwendete Quellcode ist den Listings A.2 und A.3 aufgezeigt:

Listing A.2: Konfigurationsskript der Aufnahmelogik

```
### configuration file for assos_listen
# upload
upload = False

# config for this sensor
sensorID = "al_03"

# sampling rate & chunk size
chunkSize = 1024
samplingRate = 44100 #44100 # 44100 needed for Aves sampling
# choices=[4000, 8000, 16000, 32000, 44100] :: default 16000

# sample length in seconds
sampleLength = 10

# configuration for assos_store container
ftp_server_ip = "192.168.0.157"
username = "sensor"
password = "sensor"

# storage on assos_listen device
storagePath = "/home/pi/assos_listen_pi/storage/"
```

Listing A.3: Aufnahmeroutine der Sensoren

```
import wave
import datetime
import signal
import ftplib
import sys
import os
## alsaaudio since pyaudio gave IOError:
```

```
#### [Errno Input overflowed] -9981
import alsaaudio
import numpy as np
import array
import time

# configuration for assos_listen
import config


# run the audio capture and send sound sample processes
# in parallel
from multiprocessing import Process

# CONFIG
# Set attributes: Mono, 16 bit little endian samples
FORMAT = alsaaudio.PCM_FORMAT_S16_LE
CHANNELS = 1
# read from config.py
CHUNK = config.chunkSize
RATE = config.samplingRate
RECORD_SECONDS = config.sampleLength

## HELPER FUNCTIONS
# write to ftp
def uploadFile(filename):

    print("start uploading file: " + filename)
    # connect to container
    ftp = ftplib.FTP(config.ftp_server_ip, config.username,
        ↪ config.password)

    # write file
    ftp.storbinary('STOR '+filename, open(filename, 'rb'))
    # close connection
    ftp.quit()
    print("finished uploading: " +filename)

# abort the sampling process
```

```
46 def signal_handler(signal, frame):
47     print('You pressed Ctrl+C!')
48     sys.exit(0)
49
50
51 ## MAIN FUNCTION
52 def recordAudio(recorder):
53     sampleNumber = 0
54     while (True):
55         sampleNumber = sampleNumber +1
56         print("*** recording #" + str(sampleNumber))
57
58         ## get datetime from OS
59         # %Y-%m-%d_%H%M%S_%Z is the by monitoR used
60         #  time.source formatting
61         # UTC is the default timezone
62         # ___%f is needed for localisation algorithms
63         startDateTimeStr = datetime.datetime.now().strftime("%Y-%m
               ↪ -%d_%H%M%S_UTC--%f")
64         fileName = str(config.sensorID) + "_" + startDateTimeStr +
               ↪  ".wav"
65
66         # write the wav file
67         wf = wave.open(fileName, 'w')
68         wf.setnchannels(CHANNELS)
69         wf.setsampwidth(2)
70         wf.setframerate(RATE)
71
72         for i in range(0, 1000*RECORD_SECONDS/22):
73             # write frame by frame
74             l, data = recorder.read()
75             wf.writeframes(data)
76         wf.close()
77         print(fileName + " written")
78
79         if (config.upload == True):
80             # since waiting for the upload to finish will take some
                   ↪  time
81             # and we do not want to have gaps in our sample
```

```
82          # we start the upload process in parallel
83          print("start uploading...")
84          uploadProcess = Process(target=uploadFile, args=(
               ↪ fileName,))
85          uploadProcess.start()
86
87
88
89  # ENTRYPOINT FROM CONSOLE
90  if __name__ == '__main__':
91
92      # create recorder object and load configuration
93      recorder = alsaaudio.PCM(type=alsaaudio.PCM_CAPTURE)
94      recorder.setchannels(CHANNELS)
95      recorder.setrate(RATE)
96      recorder.setformat(FORMAT)
97      recorder.setperiodsize(CHUNK)
98
99      # directory to write and read files from
100     os.chdir(config.storagePath)
101
102     # abort by pressing C
103     signal.signal(signal.SIGINT, signal_handler)
104     print('\n\n-------------------------\npress Ctrl+C to stop
           ↪ the recording')
105
106     # start recording
107     recordAudio(recorder)
```

A.4 Simulation des standartabweichungsbasierten Lokalisierungsalgorithmus

Der in Abschnitt 2.3 auf Seite 7 angesprochene Lokalisierungsalgorithmus durch Optimisierung der Standardabweichung wurde mit folgendem Python Code simuliert und getestet:

Listing A.4: Simulation des auf STD Minimierung basierten Lokalisierungsalgorithmus

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jan 6 15:17:38 2018

@author: Florian
"""

#%% IMPORTS
import pandas as pd
import os
import scipy
import numpy as np
#%%

## CONFIGURATION

# get detections from csv files
wdir = "C:/Users/Florian/Google Drive/Thesis/Aufnahme/locAlg/"
os.chdir(wdir)

sampleName = "relativeAssos"

# sensor positions ins [x,y,z]
coordsSensorA = [10,10,8]
coordsSensorB = [-8,8,-2]
coordsSensorC = [10,-10,4]
coordsSensorD = [-8,-10,6]

## ASSOS relative coordinates
coordsSensorA = [0.10,-23.10,5.99]
```

```
coordsSensorB = [-8.68,-19.48,6.03]
coordsSensorC = [-19.8,23.10,-1.28]
coordsSensorD = [19.88,2.59,-5.78]



# used to create signal data in x/y/z random position between
#  -maxSignalRandom/2 -- maxSignalRandom/2 per coordinate
maxSignalRandom = 25

# should there be added noise to the sensor detections?
noiseLevelTest = True
# noise settings
maxNoise = 1 # max noise in [m] error
# used to create random noise between 0 -- maxNoise in steps
step = 0.1

# should intermediate localization results be saved seperately
saveIntermediates = False

# bounds shall be the min/max locations from the sensor array
    ↪ plus this buffer
boundsBuffer = 20 # in units of measurement


sampleName = sampleName +"_mLevels" +str(noiseLevelTest) + "
    ↪ _mNoise" + str(maxNoise) + "_mSigRnd" + str(maxSignalRandom
    ↪ )

sensorPositions = {

                "a":coordsSensorA,
                "b":coordsSensorB,
                "c":coordsSensorC,
                "d":coordsSensorD,
              }

# save minimize steps
i = 0 # point
```

```
67 ii = 0 # step per point i
68 points = {}
69
70 ######### helper functions
71 def distance(p1,p2):
72    dist = np.linalg.norm(p1-p2) # numpy.linalg >> fastest way to
      ↪ process
73    return dist
74
75
76 ## 3d trilateration as optimization problem
77 ## localisation algorithm 'originally' by
78 # https://www.alanzucconi.com/2017/03/13/positioning-and-
   ↪ trilateration/
79 # but heavily modified by me for 1. working at all 2. working for
   ↪  3d data
80 # 3. working with pandas dataframes and therefore fast
81 # standart deviation as minimize function
82 def getSTD(startPoint, sensorLocations, sensedDistances):
83    atdDistancesToSensors = np.array([])
84    global i # point
85    global ii # step per point i
86    ii = ii + 1
87    for location, sensedDistance in zip(sensorLocations,
      ↪ sensedDistances):
88       atd_dist = distance(startPoint,location) - sensedDistance
89       atdDistancesToSensors = np.append(atdDistancesToSensors,
         ↪ atd_dist)
90    if(saveIntermediates):
91       try: # allready initialized?
92          # save minimize step for point i
93          points[i][ii] = {
94                'x':startPoint[0],
95                'y':startPoint[1],
96                'z':startPoint[2],
97                'std':atdDistancesToSensors.std()
98                }
99       except: # if not create empty dict and fill it
100         points[i] = {}
```

```
101           points[i][ii] = {
102                 'x':startPoint[0],
103                 'y':startPoint[1],
104                 'z':startPoint[2],
105                 'std':atdDistancesToSensors.std()
106                 }
107
108     return atdDistancesToSensors.std()
109
110 # initial_location: (x, y)
111 # locations: [ (x1, y1), ... ]
112 # distances: [ distanceAt1, distanceAt2, ... ]
113 def getLocation(initialLocation, sensorLocations,
    ↪ sensedDistances, bounds):
114     result = scipy.optimize.minimize(
115           getSTD,               # The error function
116           initialLocation,      # The initial guess
117                                 # Additional parameters for mse
118           args=(sensorLocations,sensedDistances,),
119           method='L-BFGS-B',    # The optimisation algorithm
120           bounds=bounds, # used bounds as (min,max)
121           options={
122                 'ftol':0.00000001,  # Tolerance
123                 'maxiter': 50000, # Maximum iterations
124                 'disp':True,    # display convergence messages
125                 })
126
127     # outcome as
128     outcome = [-99, # x
129             -99, # y
130             -99, # z
131             -99] # remainingSTD
132
133     # The optimization result represented as a OptimizeResult
        ↪ object.
134     # Important attributes are: x the solution array, success a
        ↪ Boolean flag
135     # indicating if the optimizer exited successfully and message
        ↪  which describes
```

```
136     # the cause of the termination.
137     if (result.success):
138         #print("getLocation was succesful")
139         coords = result.x # [x,y,z]
140         outcome = [coords[0], # x
141                 coords[1], # y
142                 coords[2], # z
143                 result.fun] # remainingSTD]
144     else:
145         print("getLocation failed with:")
146         print(result.message)
147 
148     return outcome
149 #####
150 
151 
152 def runAlgTest(noise,coordsSignal):
153     df_detections = pd.DataFrame(data=[coordsSignal],
154                         index=[1],
155                         columns=['signal_x','signal_y','signal_z'])
156 
157     df_sensors = pd.DataFrame.from_dict(sensorPositions,orient="
        ↪ index")
158     df_sensors.columns = ['x', 'y', 'z'] # change column names
159 
160     # calculate the distance / time it will take the signal to
        ↪ arrive at the sensor
161     # needed to find the _first encounter_ sensor and calculate
        ↪ the arrival time differences
162     def calcDistanceToSignal(row):
163         p1 = np.array([row['x'],row['y'],row['z']])
164         p2 = np.array(coordsSignal)
165         d = distance(p1,p2)
166         return d
167     df_sensors['distanceToSignal'] = df_sensors.apply(
        ↪ calcDistanceToSignal,
168                                             axis=1)
169 
170     def calcArrivalTimeDifference(row,firstEncounterTime):
```

```
171         p1 = np.array([row['x'],row['y'],row['z']])
172         p2 = np.array(coordsSignal)
173         atd = distance(p1,p2) - firstEncounterTime
174         return atd
175     # calculate the arrival time differences
176     df_sensors['arrivalTimeDifference'] = df_sensors.apply(
        ↪ calcArrivalTimeDifference,
177                 args=(np.min(df_sensors['distanceToSignal']),),
178                 axis=1)
179     df_sensors.to_csv(sampleName+'_sensorPlacement.csv', index=
        ↪ False)
180
181     # apply detection for signal
182     def setATD(row, noise):
183         for t in df_sensors.itertuples():
184             row[t.Index+'_atd'] = t.arrivalTimeDifference
185             # add noise as random[-1,..,1] * noise
186             n = (2 * np.random.random_sample((1))[0] - 1) * noise
187             row[t.Index+'_atd_n'] = t.arrivalTimeDifference + n
188             current_summed_n = 0
189             try:
190                 current_summed_n = abs(row['summed_n'])
191             except:
192                 current_summed_n = 0 # not initialized, yet
193             row['summed_n'] = current_summed_n + abs(n)
194         return row
195
196     df_detections = df_detections.apply(setATD,
197                                 args=(noise,),
198                                 axis=1)
199
200
201     ## TRYING TO GET THE LOCATION
202     sl = df_sensors[['x','y','z']].as_matrix()
203     initialLocation = np.mean(df_sensors[['x','y','z']])
204
205
206     # use bounds to get rid of huge outliers
207     bounds = (np.min(df_sensors[['x','y','z']]) - boundsBuffer,
```

```
208             np.max(df_sensors[['x','y','z']]) + boundsBuffer)
209     # np array to min / max xyz tuples
210     bounds = ((bounds[0][0],bounds[1][0]),
211             (bounds[0][1],bounds[1][1]),
212             (bounds[0][2],bounds[1][2]),)
213
214     def calcXYZ(row,bounds):
215         global i
216         i = i + 1
217
218         sensedDistances = row[['a_atd',
219                                'b_atd',
220                                'c_atd',
221                                'd_atd']]
222
223         location = getLocation(initialLocation,
224                                sl,
225                                sensedDistances.as_matrix(),
226                                bounds)
227
228         row['localized_x'] = location[0]
229         row['localized_y'] = location[1]
230         row['localized_z'] = location[2]
231         row['localized_std'] = location[3]
232
233
234         ## WITH NOIZE
235         sensedDistances = row[['a_atd_n',
236                                'b_atd_n',
237                                'c_atd_n',
238                                'd_atd_n']]
239
240         location = getLocation(initialLocation,
241                                sl,
242                                sensedDistances.as_matrix(),
243                                bounds)
244
245         row['localized_x_n'] = location[0]
246         row['localized_y_n'] = location[1]
```

```
247         row['localized_z_n'] = location[2]
248         row['localized_std_n'] = location[3]
249 
250         pointLocalized = np.array([row['localized_x'],
251                                    row['localized_y'],
252                                    row['localized_z']])
253 
254         pointLocalized_n = np.array([row['localized_x_n'],
255                                      row['localized_y_n'],
256                                      row['localized_z_n']])
257 
258 
259         pointPlayed = np.array([row['signal_x'],
260                                 row['signal_y'],
261                                 row['signal_z']])
262 
263         # calculate error distance in [m]
264         row['error_m'] = distance(pointLocalized,pointPlayed)
265         row['error_m_n'] = distance(pointLocalized_n,pointPlayed)
266 
267         return row
268 
269     df_detections = df_detections.apply(calcXYZ,args=(bounds,),
        ↪ axis=1)
270     return df_detections
271 
272 
273 
274 ### run test
275 # will hold all data
276 df_all = pd.DataFrame()
277 
278 # create test data in x/y/z random position between -
    ↪ maxSignalRandom/2 -- maxSignalRandom/2 per coordinate
279 signals = (np.random.rand(100,3) -0.5) * maxSignalRandom
280 # float range from 0.0 -- maxNoise in steps
281 noiseLevels = np.arange(0,maxNoise + step, step)
282 
283 if (noiseLevelTest):
```

```
284     for noise in noiseLevels:
285        toGo = len(signals)
286        for signal in signals:
287           toGo = toGo - 1
288           print("to go:" +str(toGo))
289           df = runAlgTest(noise,signal)
290           df_all = pd.concat([df_all,df])
291 else:
292     toGo = len(signals)
293     for signal in signals:
294           toGo = toGo - 1
295           print("to go:" +str(toGo))
296           df = runAlgTest(maxNoise,signal)
297           df_all = pd.concat([df_all,df])
298 
299 # save the dataframe to a csv file for further analysis
300 df_all.to_csv('locResults_'+sampleName+'.csv', index=False)
```

A.5 Klassifizierung der Audiodateien

Um innerhalb der aufgenommenen Audiodateien die gesuchten Vogelstrophen zu detektieren wurde folgender in R geschriebener und das package *monitoR* nutzende Code verwendet. Die über *monitoR* genutzten Templates und WAV-Trainingsdaten sind über Zenodo veröffentlicht, siehe HOEDT (2018a).

Listing A.5: R Code zur Detektion der Vogelarten in den Audioaufnahmen

```
## ASSOS ANALYSIS COMPONENT ##
# load needed libraries
library(monitoR)
library(plyr)
## CONFIG ##
setwd("C:/Users/Florian/Google Drive/Thesis/shiny/assos_analysis ")
trainingDatasets.path <- "recordings/171025/trainingFiles/edited "
recordings.path <- "recordings/171025/usableFiles"

## CREATE TEMPLATES ##
# iterate over files and create correlation templates
species <- 'cc'
score.cutoff <- 0.4

file.names <- dir(paste(trainingDatasets.path, species, sep ="/") , pattern =".wav")
first <- TRUE
for (i in 1:length(file.names))
{
  print(paste0("make corellation template for: ",file.names[i]))
  clip <- paste(trainingDatasets.path,species,file.names[i],sep =  "/")
  name <- paste0(species,"_",i)
  if (first){
    # do this once for the first template file
    first <- FALSE
    corTemplates <- makeCorTemplate(clip = clip,
                                    name = name,
                                    score.cutoff = score.cutoff,
```

```
                                select = "rectangle")
  }
  else{
    # if not first, than combine this template with the first one
    corTemplates <- combineCorTemplates(corTemplates,
                                makeCorTemplate(clip = clip,
                                                name = name,
                                                score.cutoff = score.
                                                   ↪ cutoff,
                                                select = "rectangle"))
  }
}


# save the created templates
writeCorTemplates(corTemplates, dir = "templates")

## APPLY TEMPLATES ##
## find templates for species in templates
## and create species specific templates groups
species <- c("cc", "aa", "pc")
file.names <- dir("templates", pattern =".ct")
# not filled yet
corTemplates.aa.files <- c()
corTemplates.cc.files <- c()
corTemplates.pc.files <- c()
for (ct in file.names)
{
  if(startsWith(ct,'aa'))
  {
    corTemplates.aa.files <- append(corTemplates.aa.files, ct)
  }
  else if (startsWith(ct,'cc'))
  {
    corTemplates.cc.files <- append(corTemplates.cc.files,ct)
  }
  else if (startsWith(ct,'pc'))
  {
    corTemplates.pc.files <- append(corTemplates.pc.files,ct)
```

```
  }
}

# actually read the correlatoin templates
corTemplates.aa <- readCorTemplates(files = corTemplates.aa.files,
                                    dir = "templates")
corTemplates.cc <- readCorTemplates(files = corTemplates.cc.files,
                                    dir = "templates")
corTemplates.pc <- readCorTemplates(files = corTemplates.pc.files,
                                    dir = "templates")


sensors <- c("al_01","al_02","al_03","al_04")
species <- c('aa','cc','pc')
# ITERATE OVER SENSORS
for (sensor in sensors)
{
  # ITERATE OVER SPECIES
  for (species.name in species)
  {
    if(species.name == 'aa')
    {
      corTemplates <- corTemplates.aa
    }
    else if (species.name == 'cc')
    {
      corTemplates <- corTemplates.cc
    }
    else if (species.name == 'pc')
    {
      corTemplates <- corTemplates.pc
    }

    detections.path <- paste("recordings/171025/detections",
      sensor, species.name, sep= "/")
    data.path <- paste(recordings.path, sensor, sep= "/")
```

```
102
103
104     file.names <- dir(data.path, pattern =".wav")
105
106     for(i in 1:length(file.names)){
107     #for(i in 20:80){
108       cat(sensor,species.name,"@fileNumber",i,"\n")
109       clip <- paste(data.path,file.names[i], sep = "/")
110
111       ccScore <- corMatch(clip, corTemplates,show.prog = FALSE)
112       ccPeaks <- findPeaks(ccScore)
113       ccDetections <- getDetections(ccPeaks)
114
115       if(nrow(ccDetections) > 0)
116       {
117         cat("detection for species: ", species.name,"\n")
118         newdf <- cbind(sensor,file.names[i],ccDetections)
119         write.csv(newdf,paste0(detections.path,"/",file.names[i],".
              ↪ csv"))
120       }
121     }
122   }
123 }
124
125
126 ### python and pandas used to find matching detections
127 ### and apply played data info and loudspeaker position
128 ### and orientation to it
```

A.6 Ermittlung des Azimuth

Um die in Abschnitt 1.3 auf Seite 3 genannte These 2.c zu prüfen wird die Ausrichtung des Lautsprechers im Verhältnis zu den Sensorpositionen ermittelt. Die Berechnung des Azimuth wurde hierbei über die PostGIS Funktion *ST_Azimuth* durchgeführt, wie in folgenden Listing gezeigt:

Listing A.6: PostGIS Code für die Berechnung des Azimuth zwischen Aufnahmeposition und Richtung und den jeweiligen Sensoren

```
1  UPDATE samplepoints_xyz
2  SET
3  azimuth_a = degrees(
4        st_azimuth(geom_single,
5              (SELECT geom_single
6                    FROM samplepoints_xyz
7                    WHERE name = 'A'))),
8  azimuth_b = degrees(
9        st_azimuth(geom_single,
10             (SELECT geom_single
11                   FROM samplepoints_xyz
12                   WHERE name = 'B'))),
13 azimuth_c = degrees(
14       st_azimuth(geom_single,
15             (SELECT geom_single
16                   FROM samplepoints_xyz
17                   WHERE name = 'C'))),
18 azimuth_d = degrees(
19       st_azimuth(geom_single,
20             (SELECT geom_single
21                   FROM samplepoints_xyz
22                   WHERE name = 'D'))),
23 azimuth_al_01 = degrees(
24       st_azimuth(geom_single,
25             (SELECT geom_single
26                   FROM samplepoints_xyz
27                   WHERE name = 'al_01'))),
28 azimuth_al_02 = degrees(
29       st_azimuth(geom_single,
30             (SELECT geom_single
```

```
31                  FROM samplepoints_xyz
32                  WHERE name = 'al_02'))),
33 azimuth_al_03 = degrees(
34       st_azimuth(geom_single,
35             (SELECT geom_single
36                  FROM samplepoints_xyz
37                  WHERE name = 'al_03'))),
38 azimuth_al_04 = degrees(
39       st_azimuth(geom_single,
40             (SELECT geom_single
41                  FROM samplepoints_xyz
42                  WHERE name = 'al_04')))
```

A.7 Ermittlung der schallverschattenden Objekte

Um die in Abschnitt 1.3 auf Seite 3 genannte These 2.a zu prüfen wird von der Schallgeberposition ein Schallkegel zu den jeweiligen Mikrofonen konstruiert (als 3d Feature) und ein 3d-Intersect mit der Punktwolke des Laserscans wie in Abschnitt 3.5 auf Seite 23 beschrieben durchgeführt. Die Anzahl der überschnittenen Punkte entspricht dem Maß der Schallverschattung. Der hierfür verwendete PostGIS Code ist in folgendem Listing aufgezeigt:

Listing A.7: PostGIS Code für die Ermittlung der schallverschattenden Objekte

```
1  -- IMPORT DSM data from csv file
2  DROP TABLE IF EXISTS dsm;
3
4  CREATE TABLE dsm(
5    easting float,
6   northing float,
7     height float
8  );
9
10 COPY dsm ( easting, northing, height )
11    FROM 'C:/temp/dom1l-fp_32525_5735_1_nw.xyz'
12    WITH
13         DELIMITER AS ','
14         CSV HEADER;
15
16 SELECT AddGeometryColumn('public', 'dsm', 'geom', 25832, 'POINT',
       ↪  3);
17
18 UPDATE dsm SET dsm_aoi.geom = ST_SetSRID(ST_MakePoint(easting,
       ↪ northing,height),25832);
19
20 CREATE INDEX dsm_geom_gist ON dsm USING GIST (geom);
21
22 -- CLIP DSM DATA to get aoi dataset
23 CREATE TABLE dsm_aoi
24 AS
25 SELECT * FROM dsm
26 WHERE ST_WITHIN(geom,
```

```
                            (SELECT ST_SetSRID(ST_Expand(
                                ST_Extent(geom), 50),25832)
                                FROM samplepoints)
);


--- CREATE SAMPLELINES
DROP TABLE IF EXISTS samplelines;

CREATE TABLE samplelines
(
    id serial,
    name character varying(50) COLLATE pg_catalog."default",
    CONSTRAINT samplelines_pkey PRIMARY KEY (id)
);

SELECT AddGeometryColumn('public', 'samplelines', 'geom', 25832,
    ↪ 'LINESTRING', 3);

INSERT INTO samplelines (geom, name)
-- s1
      SELECT
            ST_MAKELINE(
                  (SELECT geom
                        FROM samplepoints_xyz
                        WHERE name = 'al_01'),
                  (SELECT geom
                        FROM samplepoints_xyz
                        WHERE name = 'sound_01')) AS geom,
            's1' AS name
      UNION
      SELECT
            ST_MAKELINE(
                  (SELECT geom
                        FROM samplepoints_xyz
                        WHERE name = 'al_02'),
                  (SELECT geom
                        FROM samplepoints_xyz
                        WHERE name = 'sound_01')) AS geom,
```

```
65              's1' AS name
66        UNION
67        SELECT
68              ST_MAKELINE(
69                    (SELECT geom
70                          FROM samplepoints_xyz
71                          WHERE name = 'al_03'),
72                    (SELECT geom
73                          FROM samplepoints_xyz
74                          WHERE name = 'sound_01')) AS geom,
75              's1' AS name
76        UNION
77        SELECT
78              ST_MAKELINE(
79                    (SELECT geom
80                          FROM samplepoints_xyz
81                          WHERE name = 'al_04'),
82                    (SELECT geom
83                          FROM samplepoints_xyz
84                          WHERE name = 'sound_01')) AS geom,
85              's1' AS name
86        -- END
87  -- s5
88        UNION
89        SELECT
90              ST_MAKELINE(
91                    (SELECT geom
92                          FROM samplepoints_xyz
93                          WHERE name = 'al_01'),
94                    (SELECT geom
95                          FROM samplepoints_xyz
96                          WHERE name = 'sound_05')) AS geom,
97              's5' AS name
98        UNION
99        SELECT
100             ST_MAKELINE(
101                   (SELECT geom
102                         FROM samplepoints_xyz
103                         WHERE name = 'al_02'),
```

```
104                   (SELECT geom
105                         FROM samplepoints_xyz
106                         WHERE name = 'sound_05')) AS geom,
107             's5' AS name
108       UNION
109       SELECT
110             ST_MAKELINE(
111                   (SELECT geom
112                         FROM samplepoints_xyz
113                         WHERE name = 'al_03'),
114                   (SELECT geom
115                         FROM samplepoints_xyz
116                         WHERE name = 'sound_05')) AS geom,
117             's5' AS name
118       UNION
119       SELECT
120             ST_MAKELINE(
121                   (SELECT geom
122                         FROM samplepoints_xyz
123                         WHERE name = 'al_04'),
124                   (SELECT geom
125                         FROM samplepoints_xyz
126                         WHERE name = 'sound_05')) AS geom,
127             's5' AS name
128       --END
129 -- s6
130       UNION
131       SELECT
132             ST_MAKELINE(
133                   (SELECT geom
134                         FROM samplepoints_xyz
135                         WHERE name = 'al_01'),
136                   (SELECT geom
137                         FROM samplepoints_xyz
138                         WHERE name = 'sound_06')) AS geom,
139             's6' AS name
140       UNION
141       SELECT
142             ST_MAKELINE(
```

```
143                   (SELECT geom
144                         FROM samplepoints_xyz
145                         WHERE name = 'al_02'),
146                   (SELECT geom
147                         FROM samplepoints_xyz
148                         WHERE name = 'sound_06')) AS geom,
149             's6' AS name
150       UNION
151       SELECT
152             ST_MAKELINE(
153                   (SELECT geom
154                         FROM samplepoints_xyz
155                         WHERE name = 'al_03'),
156                   (SELECT geom
157                         FROM samplepoints_xyz
158                         WHERE name = 'sound_06')) AS geom,
159             's6' AS name
160       UNION
161       SELECT
162             ST_MAKELINE(
163                   (SELECT geom
164                         FROM samplepoints_xyz
165                         WHERE name = 'al_04'),
166                   (SELECT geom
167                         FROM samplepoints_xyz
168                         WHERE name = 'sound_06')) AS geom,
169             's6' AS name
170       --END
171 -- s7
172       UNION
173       SELECT
174             ST_MAKELINE(
175                   (SELECT geom
176                         FROM samplepoints_xyz
177                         WHERE name = 'al_01'),
178                   (SELECT geom
179                         FROM samplepoints_xyz
180                         WHERE name = 'sound_07')) AS geom,
181             's7' AS name
```

```
182       UNION
183       SELECT
184             ST_MAKELINE(
185                   (SELECT geom
186                         FROM samplepoints_xyz
187                         WHERE name = 'al_02'),
188                   (SELECT geom
189                         FROM samplepoints_xyz
190                         WHERE name = 'sound_07')) AS geom,
191             's7' AS name
192       UNION
193       SELECT
194             ST_MAKELINE(
195                   (SELECT geom
196                         FROM samplepoints_xyz
197                         WHERE name = 'al_03'),
198                   (SELECT geom
199                         FROM samplepoints_xyz
200                         WHERE name = 'sound_07')) AS geom,
201             's7' AS name
202       UNION
203       SELECT
204             ST_MAKELINE(
205                   (SELECT geom
206                         FROM samplepoints_xyz
207                         WHERE name = 'al_04'),
208                   (SELECT geom
209                         FROM samplepoints_xyz
210                         WHERE name = 'sound_07')) AS geom,
211             's7' AS name
212       --END
213 -- s8
214       UNION
215       SELECT
216             ST_MAKELINE(
217                   (SELECT geom
218                         FROM samplepoints_xyz
219                         WHERE name = 'al_01'),
220                   (SELECT geom
```

```
221                         FROM samplepoints_xyz
222                         WHERE name = 'sound_08')) AS geom,
223             's8' AS name
224       UNION
225       SELECT
226             ST_MAKELINE(
227                   (SELECT geom
228                         FROM samplepoints_xyz
229                         WHERE name = 'al_02'),
230                   (SELECT geom
231                         FROM samplepoints_xyz
232                         WHERE name = 'sound_08')) AS geom,
233             's8' AS name
234       UNION
235       SELECT
236             ST_MAKELINE(
237                   (SELECT geom
238                         FROM samplepoints_xyz
239                         WHERE name = 'al_03'),
240                   (SELECT geom
241                         FROM samplepoints_xyz
242                         WHERE name = 'sound_08')) AS geom,
243             's8' AS name
244       UNION
245       SELECT
246             ST_MAKELINE(
247                   (SELECT geom
248                         FROM samplepoints_xyz
249                         WHERE name = 'al_04'),
250                   (SELECT geom
251                         FROM samplepoints_xyz
252                         WHERE name = 'sound_08')) AS geom,
253             's8' AS name
254       --END
255 ;
256
257 --- SELECT LIDAR POINTS BY POSITION
258 DROP TABLE IF EXISTS obstacles;
259
```

```
CREATE TABLE obstacles
(
    id serial,
    name character varying(50) COLLATE pg_catalog."default",
    CONSTRAINT obstacles_pkey PRIMARY KEY (id)
);

SELECT AddGeometryColumn('public', 'obstacles', 'geom', 25832, '
    ↪ POINT', 3);

INSERT INTO obstacles (geom, name)
       SELECT dsm_aoi.geom, samplelines.name
       FROM dsm_aoi, samplelines
       WHERE
       samplelines.name = 's1' AND
       ST_3DDistance(samplelines.geom ,dsm_aoi.geom) < 0.5
       UNION
       SELECT dsm_aoi.geom, samplelines.name
       FROM dsm_aoi, samplelines
       WHERE
       samplelines.name = 's4' AND
       ST_3DDistance(samplelines.geom ,dsm_aoi.geom) < 0.5
       UNION
       SELECT dsm_aoi.geom, samplelines.name
       FROM dsm_aoi, samplelines
       WHERE
       samplelines.name = 's5' AND
       ST_3DDistance(samplelines.geom ,dsm_aoi.geom) < 0.5
       UNION
       SELECT dsm_aoi.geom, samplelines.name
       FROM dsm_aoi, samplelines
       WHERE
       samplelines.name = 's6' AND
       ST_3DDistance(samplelines.geom ,dsm_aoi.geom) < 0.5
       UNION
       SELECT dsm_aoi.geom, samplelines.name
       FROM dsm_aoi, samplelines
       WHERE
       samplelines.name = 's7' AND
```

```
298         ST_3DDistance(samplelines.geom ,dsm_aoi.geom) < 0.5
299         UNION
300         SELECT dsm_aoi.geom, samplelines.name
301         FROM dsm_aoi, samplelines
302         WHERE
303         samplelines.name = 's8' AND
304         ST_3DDistance(samplelines.geom ,dsm_aoi.geom) < 0.5
305 ;
306
307 --- GET COUNTS PER sample
308 SELECT name, count(*)
309         FROM obstacles
310         GROUP BY name;
```

A.8 Lokalisierung der Aufnahmedaten

Um die eigentliche Lokalisation durchzuführen sind diverse Zwischenschritte erforderlich. So sind die relativen Zeitstempel der Detektionen zu absoluten Zeitstempeln zu verrechnen, die gemessenen Temperaturen je Detektion zu ermitteln. Darüberhinaus gilt es das selbe Ereignis in den jeweiligen Audiospuren der Sensoren zu detektieren, um die delta TDOA zu ermitteln. Im anschließenden Teil des verwendeten Python codes wird die Lokalisierung über *Scipy.Minimize* durchgeführt. Zuletzt werden die über PostGIS ermittelten Zusatzinformationen (Deltawinkel, LIDAR Return Point count) an den Datensatz angehangen.

Listing A.8: Analyse der aus monitoR errechneten Detektionen über Python

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 21 15:36:43 2017

@author: Florian
"""

######## working with the detection files
#%% IMPORTS
import glob
import pandas as pd
import datetime
import os
import scipy
import numpy as np

#%% SETUP
# get detections from csv files
wdir = "C:/Users/Florian/Google Drive/Thesis/shiny/assos_
    ↪ analysis/recordings/171025/"
detectionsPath = os.path.join(wdir,"detections")
analysisPath = os.path.join(wdir,"analysis")
#%% LOADING CSV FILES
# iterate recursive over detections path and get a list of csv
    ↪ files
csvFileList = glob.glob(detectionsPath+'/**/**/*.csv')
```

```
# read those csv files into panda dataframes (df)
df_from_each_file = (pd.read_csv(f) for f in csvFileList)
# concat all dfs to one huge df
df_detections = pd.concat(df_from_each_file, ignore_index=True)

# date.time value from monitoR is wrong - therefore drop it!
df_detections = df_detections.drop('date.time',axis=1)

#%% PROCESS DETECTION FILES
# get file.names datetime from the recordings start
# + relative timedelta of the detection by monitoR in seconds from the samples
# start to get the UTC sensed time of the detection per row
def calcSensedTime(row):
    timeStart = datetime.datetime.strptime(row['file.names[i]'][6:-4],
                                           "%Y-%m-%d_%H%M%S_UTC--%f")
    timeDetection = timeStart + datetime.timedelta(seconds = row['time'])
    return timeDetection

# apply a new row "sensedTime" as function calcSensedTime per row
df_detections['sensedTime'] = df_detections.apply(calcSensedTime, axis=1)

# index the detections by sensed time as timeseries
df_detections_indexed = df_detections.set_index(['sensedTime'])


# get sensor name from file.name by subsetting the string
# per row of dataframe
def getSensor(row):
    sensorName = row['file.names[i]'][:5]
    return sensorName

# apply new row "sensor" as function of getSensor per row
df_detections['sensor'] = df_detections.apply(getSensor,axis=1)

```

```
#%%
##########################################
# load played files as dataframe
playedPath = os.path.join(wdir,"playedFiles")
df_played = pd.read_csv(os.path.join(playedPath,"played_vitro.csv
    ↪ "),
                        delimiter=";")

def getDatetime(row):
    timeString = str(row['TIME']) # convert to string for easy
        ↪ subsetting
    h = timeString[0:2]
    m = timeString[2:4]
    s = timeString[4:6]
    # 2017-10-25 as survey date
    calcedDatetime = datetime.datetime(year = 2017, month = 10,
        ↪ day = 25,
                                       hour = int(h),
                                       minute = int(m),
                                       second = int(s))
    return calcedDatetime

df_played['datetime'] = df_played.apply(getDatetime,axis=1)
# index the played samples by time as timeseries
df_played_indexed = df_played.set_index(['datetime'])

#%%
##########################################
# load temperature data as df
df_temperatures = pd.read_csv(analysisPath+'/temperatures.csv')

def getDatetime(row):
    time = datetime.datetime.strptime(row['datetime'],"%d.%m.%Y %
        ↪ H:%M") # MESZ
    time = time - datetime.timedelta(hours = 2) # converting to
        ↪ UTC from MESZ
    return time

# parse time values from datetimestring and use those as index
```

```
93 df_temperatures['time'] = df_temperatures.apply(getDatetime,axis
      ↪ =1)
94 df_temperatures = df_temperatures.set_index(['time'])
95 # drop not needed columns
96 df_temperatures = df_temperatures.drop('datetime',1)
97
98 #%%
99 ##########################################
100 # get matching detections per species/sensor/time
101 # iterate over all detections and find matching pairs between
      ↪ the different sensors
102 # for the ~same time frame and species
103
104 # the maximum deltatime allowed to count as one matching
      ↪ detection
105 # format hh:mm:ss.ff
106 # speed of sound @ 15C ~ 340,3 m/s
107 # farthest distance between two sensors is ~ 50,3 m
108 # which relates to a maximum delta time of ~ 0,15 seconds
109 maxDeltaTime = pd.Timedelta('00:00:00.15')
110
111 matchingDetections = {}
112 errorDetections = {}
113 index_matchingDetections = 0
114
115 toDo = len(df_detections_indexed)
116 errors = []
117
118 for index, row in df_detections_indexed.iterrows():
119     index_matchingDetections = index_matchingDetections + 1
120
121
122     toDo = toDo - 1
123     print("todo :: " + str(toDo))
124
125     sensedTime = index
126     # not only search for species (eg. Alauda arvensis)
127     # but also for the exact correlation template match!
128     # otherwise there are loads of false positives!
```

```
129     species = row['template']
130     sensor = row['sensor']
131 
132     # get played species and direction
133     nearestPlayed = df_played_indexed.iloc[df_played_indexed.index
           ↪ .get_loc(sensedTime,method='nearest')]
134 
135     # get temperature
136     nearestTemperature = df_temperatures.iloc[df_temperatures.
           ↪ index.get_loc(sensedTime,method='nearest')]
137 
138     # holds sensors with should be iterated
139     sensorsToFetch = ["al_01","al_02","al_03","al_04"]
140     # pop the current sensor out of the list
141     sensorsToFetch.pop(sensorsToFetch.index(sensor))
142 
143     # append this detection to the list
144     detection = {## sensed info
145               "sensedSpecies":species,
146               "masterDetectionFile": row['file.names[i]'],
147               "masterDetectionTime":sensedTime,
148               "masterDetectionTime_sinceFileStart":row['time'],
149               sensor+"_detectionScore":row['score'],
150               "temperature":nearestTemperature['temp'],
151 
152               ## the detection score will be summed up later
153               "summed_detectionScore":row['score'],
154 
155               ## actually played species and position
156               ## and facing direction of the loudspeaker
157               "playedSpecies":nearestPlayed['SPECIES'],
158               "playedAtPoint":nearestPlayed['POINT'],
159               "direction":nearestPlayed['DIRECTION'],
160               sensor:0} # master sensor detection is 0
161                       # because we use timedeltas
162 
163     ## fill in the master sensors info for convenience
164     detection[sensor+"_matchedFile"] = row['file.names[i]']
165     detection[sensor+"_sinceFileStart"] = row['time']
```

```
166
167     ## subsetting per sensor and species
168     for sensorName in sensorsToFetch:
169         # find matching detection in data from sensor:: sensorName
170         # define mask
171         mask = (df_detections_indexed["template"] == species) & (
                ↪ df_detections_indexed["sensor"] == sensorName)
172         # subset the dataframe by mask
173         subset = df_detections_indexed[mask]
174
175         # check if there are possible candidates
176         if (len(subset) > 0):
177             # drop duplicate timestamps for this sensor and species
178             # except for the first instance
179             subset = subset[~subset.index.duplicated(keep='first')]
180             # resort the index
181             subset = subset.sort_index()
182
183             # store index as row, otherwise it will get lost by the
                    ↪  iloc command
184             subset['sensedTime'] = subset.index
185
186             # fetch the nearest datapoint from sensedTime of row (
                    ↪ outer loop)
187             nearestDetection = subset.iloc[subset.index.get_loc(
                    ↪ sensedTime,method='nearest')]
188
189             # calculate timedelta between sensed time of master
                    ↪ sensor
190             # and the nearest detection on one of the slave sensors
191             tDelta = sensedTime - nearestDetection['sensedTime']
192
193             if (abs(tDelta) < maxDeltaTime ):
194                 # add sensors detection as relative timedelta
195                 # to the detection dict
196                 detection[sensorName+"_matchedFile"] =
                        ↪ nearestDetection['file.names[i]']
197                 detection[sensorName+"_sinceFileStart"] =
                        ↪ nearestDetection['time']
```

```
198             detection[sensorName+"_detectionScore"] =
                    ↪ nearestDetection['score']
199             detection[sensorName] = tDelta.total_seconds()
200             detection["summed_detectionScore"] = detection["
                    ↪ summed_detectionScore"] + nearestDetection['
                    ↪ score']
201         else:
202             message = row['file.names[i]'] + " :: deltatime is
                    ↪ to big :: " +str(abs(tDelta)) + " :: detection
                    ↪ will be discarded"
203             errorDetections[index_matchingDetections] =
                    ↪ detection
204             errors.append(message)
205             print(message)
206
207     else:
208         #print("there is no detection for [" + species + "|"+
                ↪ sensorName +"] -- len:" + str(len(subset)))
209         errors.append(row['file.names[i]'] + " ::there is no
                ↪ detection for [" + species + "|"+ sensorName +"]
                ↪ -- len:" + str(len(subset)))
210         errorDetections[index_matchingDetections] = detection
211
212     # add detection to matching detections
213     matchingDetections[index_matchingDetections] = detection
214
215 if (len(errors) > 0):
216     print("finished with " + str(len(errors)) + " errors")
217     #print(errors)
218 else:
219     print("finished without errors")
220
221 #%%
222 # format matching detections dict to pandas dataframe
223 df_matchingDetections = pd.DataFrame.from_dict(
        ↪ matchingDetections,orient="index")
224 df_errorDetections = pd.DataFrame.from_dict(errorDetections,
        ↪ orient="index")
225
```

```
226 # save the raw file for later analysis
227 df_matchingDetections.to_csv(analysisPath+'/matching_detections_
    ↪ raw.csv', index=False)
228 df_errorDetections.to_csv(analysisPath+'/error_detections.csv',
    ↪ index=False)
229
230 # create a cleaned dataset for localisation algorithm
231 # clean duplicate
232 df_matchingDetections_cleaned = df_matchingDetections.drop_
    ↪ duplicates()
233 # drop rows with NAN values (not matched by every microphone)
234 df_matchingDetections_cleaned = df_matchingDetections_cleaned.
    ↪ dropna()
235
236 # write out the dataframes to csv
237 df_matchingDetections_cleaned.to_csv(analysisPath+'/matching_
    ↪ detections_cleaned_withSummedScores.csv', index=False)
238
239 df_matchingDetections.to_csv(analysisPath+'/matching_detections.
    ↪ csv', index=False)
240
241 #%% LOAD PROCESSED MATCHING DETECTIONS
242 df_matchingDetections_cleaned = pd.read_csv(analysisPath+'/
    ↪ matching_detections_cleaned_withSummedScores.csv')
243 df_sensorLocations = pd.read_csv(analysisPath+'/sensor_locations.
    ↪ csv')
244
245 # as starting point for the optimization algorithm
246 # calculate the 3d centroid as mean of all sensor locations
247 initialLocation = np.mean(df_sensorLocations)
248
249 #%%
250
251 # The speed of sound is calculated from an appropriate
    ↪ temperature value T
252 # by v = 331.3 * sqrt(1 + T / 273.15)
253 def travelledDistance(row,sensor):
254     timedelta = row[sensor]
255     velocity = 331.3 * np.sqrt((1 + row['temperature']/ 273.15))
```

```
256     travelledDistance = timedelta * velocity
257     return abs(travelledDistance)
258 
259 # apply new row "dn" as function of travelledDistance per row
260 for i in range(1,5):
261     distance = 'd' + str(i)
262     sensor = 'al_0' + str(i)
263     df_matchingDetections_cleaned[distance] = df_
            ↪ matchingDetections_cleaned.apply(travelledDistance,
264                                        args=(sensor,),
265                                        axis=1)
266 
267 df_matchingDetections_cleaned.to_csv(analysisPath+'/matching_
        ↪ detections_cleaned_calced_distances_withSummedScores.csv',
        ↪ index=False)
268 
269 #%% 3d trilateration as optimization problem
270 ## localisation algorithm 'originally' by
271 # https://www.alanzucconi.com/2017/03/13/positioning-and-
        ↪ trilateration/
272 # but heavily modified by me for 1. working at all 2. working for
        ↪  3d data
273 # 3. working with pandas dataframes and therefore fast
274 
275 
276 # p1 = np.array([0,0,0])
277 # p2 = np.array([100,0,0])
278 # result 100.0
279 def distance3d(p1,p2):
280     dist = np.linalg.norm(p1-p2) # numpy.linalg >> fastest way to
            ↪  process
281     return dist
282 
283 # save minimize steps
284 #i = 0
285 #points = {}
286 
287 # mean square error as minimize function
288 def getSTD(startPoint, sensorLocations, sensedDistances):
```

```
289     atdDistancesToSensors = np.array([])
290     #global i
291     #i = i + 1
292     for location, distance in zip(sensorLocations,
           ↪ sensedDistances):
293         atd_dist = distance3d(startPoint,location) - distance
294         atdDistancesToSensors = np.append(atdDistancesToSensors,
               ↪ atd_dist)
295    # save minimize steps
296    # points[i] = {'x':startPoint[0],
297    #           'y':startPoint[1],
298    #           'z':startPoint[2],
299    #           'std':atdDistancesToSensors.std()
300    #          }
301     return atdDistancesToSensors.std()
302
303 # initial_location: (x, y)
304 # locations: [ (x1, y1), ... ]
305 # distances: [ distanceAt1, distanceAt2, ... ]
306 def getLocation(initialLocation, sensorLocations,
       ↪ sensedDistances, bounds):
307     result = scipy.optimize.minimize(
308           getSTD,               # The error function
309           initialLocation,      # The initial guess
310                                 # Additional parameters for mse
311           args=(sensorLocations,sensedDistances,),
312           method='L-BFGS-B',    # The optimisation algorithm
313           bounds=bounds, # used bounds as (min,max)
314           options={
315                 'ftol':0.00000001,  # Tolerance
316                 'maxiter': 50000, # Maximum iterations
317                 'disp':True,    # display convergence messages
318                 })
319
320     # outcome as
321     outcome = [-99, # x
322             -99, # y
323             -99, # z
324             -99] # remainingSTD
```

```
325
326     # The optimization result represented as a OptimizeResult
            ↪ object.
327     # Important attributes are: x the solution array, success a
            ↪ Boolean flag
328     # indicating if the optimizer exited successfully and message
            ↪  which describes
329     # the cause of the termination.
330     if (result.success):
331         coords = result.x # [x,y,z]
332         outcome = [coords[0], # x
333                 coords[1], # y
334                 coords[2], # z
335                 result.fun] # remainingSTD]
336     else:
337         print("getLocation failed with:")
338         print(result.message)
339
340     return outcome
341
342 #%% TRYING TO GET THE LOCATION
343 sl = df_sensorLocations.as_matrix()
344 initialLocation = np.mean(df_sensorLocations)
345
346 # bounds shall be the min/max locations from the sensor array
        ↪ plus this buffer
347 boundsBuffer = 30 # in units of measurement
348 # use bounds to hopefully get rid of huge outliers
349 bounds = (np.min(df_sensorLocations) - boundsBuffer, np.max(df_
        ↪ sensorLocations) + boundsBuffer)
350 # np array to min / max xyz tuples
351 bounds = ((bounds[0][0],bounds[1][0]),
352         (bounds[0][1],bounds[1][1]),
353         (bounds[0][2],bounds[1][2]),)
354
355 def calcXYZ(row,bounds):
356     print("index:" + str(row.name))
357     sensedDistances = row[['d1','d2','d3','d4']]
358
```

```
359     location = getLocation(initialLocation,sl,sensedDistances.as_
            ↪ matrix(),
360                            bounds)
361     row['localized_x'] = location[0]
362     row['localized_y'] = location[1]
363     row['localized_z'] = location[2]
364     row['localized_std'] = location[3]
365     return row
366 
367 df = df_matchingDetections_cleaned
368 
369 df = df.apply(calcXYZ,args=(bounds,), axis=1)
370 
371 df.to_csv(analysisPath+'/localized_detections_full_attributes_
        ↪ bounded_all_withSummedScores.csv', index=False)
372 
373 #
        ↪ ==========================================================================
        ↪
374 # the single steps could be saved via:
375 # df_optimizationSteps = pd.DataFrame.from_dict(
        ↪ optimizationSteps,orient='index')
376 # df_optimizationSteps.to_csv(analysisPath+'/optimizationSteps.
        ↪ csv', index=False)
377 #
378 #
        ↪ ==========================================================================
        ↪
379 
380 #%% ANALYZE RESULTS
381 # read needed data as panda dfs
382 df_localized = df #pd.read_csv(analysisPath+'/localized_
        ↪ detections_full_attributes_bounded_all.csv')
383 df_sensorLocations = pd.read_csv(analysisPath+'/sensor_locations.
        ↪ csv')
384 df_setup = pd.read_csv(analysisPath+'/setup_sensors_speaker.csv')
        ↪
385 df_bearings = pd.read_csv(analysisPath+'/Bearing.csv')
```

```
386 df_obstacles = pd.read_csv(analysisPath+'/ObstacleCountPerSample.
    ↪ csv')
387
388 def appendAnalysisColumns(df):
389     # select pointPlayed info and join it to the df
390     p = df['playedAtPoint']
391     if (p < 5): # samples 1-4 are played at position 1
392         p = 1
393     sound = "sound_0" + str(p)
394
395     # define mask for location error
396     mask = (df_setup['name'] == sound)
397     # subset the dataframe by mask
398     subset = df_setup[mask]
399
400     # save point info
401     df['played_x'] = subset['point_x'].values[0]
402     df['played_y'] = subset['point_y'].values[0]
403     df['played_z'] = subset['point_z'].values[0]
404
405     # instantiate points as np.arrays
406     pointLocalized = np.array([df['localized_x'],
407                                df['localized_y'],
408                                df['localized_z']])
409
410
411     pointPlayed = np.array([df['played_x'],
412                             df['played_y'],
413                             df['played_z']])
414
415     # calculate error distance in [m]
416     df['error_m'] = distance3d(pointLocalized,pointPlayed)
417
418     # define mask for bearings
419     mask = (df_bearings['name'] == sound)
420     # subset the dataframe by mask
421     subset = df_bearings[mask]
422
423     # join summed azimuth value to dataframe
```

```
424     df['summed_azimuth'] = subset['SUM_to_' + df['direction']].
           ↪ values[0] # pick corrosponding azimuth sum
425
426
427     # join obstacle count to dataset
428     obstacleCount = df_obstacles[df_obstacles["playedAtPoint"] ==
           ↪  p]['count'].values[0]
429     df['obstacleCount'] = obstacleCount
430
431     return df
432
433
434 df_localized = df_localized.apply(appendAnalysisColumns,axis=1)
435 df_localized.to_csv(analysisPath+'/localized_detections_analyzed.
       ↪ csv', index=False)
```

A.9 Temperaturmessungen

Die Temperatur ist ein Parameter zur Berechnung der Schallgeschwindigkeit. Die Temperaturdaten dieser Arbeit wurden auf dem Campus der Hochschule an der Wetterstation Höxter aufgenommen:

Tabelle A.1: Die an der Wetterstation Höxter ($51° \; 46' \; 3.36'' N, \;\; 9° \; 22' \; 9,86'' E$, 156m über NN) gemessenen Temperaturen zur Aufnahmezeit.

Datetime	**temperature [C°]**	**Datetime** (cont.)	**temperature [C°]** (cont.)
25.10.2017 15:00	16.3	25.10.2017 17:40	14.3
25.10.2017 15:10	16.3	25.10.2017 17:50	14.3
25.10.2017 15:20	16.2	25.10.2017 18:00	14.2
25.10.2017 15:30	16.1	25.10.2017 18:10	14.1
25.10.2017 15:40	16.0	25.10.2017 18:20	14.1
25.10.2017 15:50	16.1	25.10.2017 18:30	14.1
25.10.2017 16:00	15.9	25.10.2017 18:40	14.0
25.10.2017 16:10	15.7	25.10.2017 18:50	14.1
25.10.2017 16:20	15.5	25.10.2017 19:00	13.9
25.10.2017 16:30	15.3	25.10.2017 19:10	13.8
25.10.2017 16:40	15.1	25.10.2017 19:20	13.8
25.10.2017 16:50	14.9	25.10.2017 19:30	13.8
25.10.2017 17:00	14.8	25.10.2017 19:40	13.6
25.10.2017 17:10	14.8	25.10.2017 19:50	13.6
25.10.2017 17:20	14.7	25.10.2017 20:00	13.6
25.10.2017 17:30	14.6	—	—

A.10 Abgespielte Audiodateien

Während der Datenaufnahme sind die Zeitstempel der jeweils abgespielten Audiosamples nach Spezies dokumentiert worden wie in nachfolgender Tabelle gezeigt. Die Aufnahmepunkte 1–4 sind lageidentisch, daher wurden diese zu Punkt 1 zusammengefasst.

Tabelle A.2: Während der Aufnahme abgespielte Audiodateien. Der Zeitstempel ist in hhmmss codiert.

Time	Point	Direction	Species	Time (cont.)	Point (cont.)	Direction (cont.)	Species (cont.)
152223	1	A	aa	155750	6	B	pc
152232	1	A	aa	155755	6	B	pc
152340	1	A	aa	155804	6	B	pc
152409	1	A	cc	155806	6	B	pc
152438	1	A	cc	155844	6	C	aa
152447	1	A	cc	155854	6	C	aa
152457	1	A	cc	155953	6	C	cc
152507	1	A	cc	160021	6	C	cc
152517	1	A	cc	160032	6	C	cc
152523	1	A	pc	160035	6	C	cc
152526	1	A	pc	160038	6	C	cc
152536	1	A	pc	160043	6	C	cc
152549	1	A	pc	160047	6	C	cc
152555	1	A	pc	160052	6	C	cc
152728	1	B	cc	160058	6	C	pc
152732	1	B	cc	160129	6	C	pc
152739	1	B	cc	160452	6	D	aa
152745	1	B	cc	160500	6	D	aa
152752	1	B	cc	160502	6	D	aa
152755	1	B	cc	160512	6	D	aa
152800	1	B	cc	160528	6	D	aa
152804	1	B	cc	160531	6	D	aa
152811	1	B	cc	160541	6	D	cc

Time	Point	Direction	Species	Time (cont.)	Point (cont.)	Direction (cont.)	Species (cont.)
152815	1	B	cc	160545	6	D	cc
152821	1	B	cc	160553	6	D	cc
152824	1	B	cc	160557	6	D	cc
152827	1	B	cc	160600	6	D	cc
152919	1	B	pc	160609	6	D	cc
153000	1	D	aa	160613	6	D	cc
153007	1	D	aa	160618	6	D	cc
153015	1	D	aa	160621	6	D	cc
153046	1	D	aa	160630	6	D	cc
153056	1	D	cc	160636	6	D	cc
153101	1	D	cc	160640	6	D	cc
153109	1	D	cc	160643	6	D	cc
153113	1	D	cc	160648	6	D	cc
153115	1	D	cc	160652	6	D	cc
153119	1	D	cc	160658	6	D	cc
153125	1	D	cc	160703	6	D	pc
153128	1	D	cc	160728	6	D	pc
153134	1	D	cc	160817	7	A	aa
153135	1	D	cc	160826	7	A	aa
153145	1	D	cc	160835	7	A	aa
153151	1	D	cc	160845	7	A	aa
153155	1	D	cc	160933	7	A	cc
153158	1	D	cc	160943	7	A	cc
153204	1	D	cc	160953	7	A	cc
153207	1	D	cc	161025	7	A	cc
153214	1	D	cc	161032	7	A	cc
153218	1	D	pc	161037	7	A	pc
153224	1	D	pc	161042	7	A	pc
153229	1	D	pc	161047	7	A	pc
153234	1	D	pc	161053	7	A	pc
153245	1	D	pc	161103	7	A	pc
153327	1	C	aa	161935	7	B	aa

Time	Point	Direction	Species	Time (cont.)	Point (cont.)	Direction (cont.)	Species (cont.)
153331	1	C	aa	161944	7	B	aa
153343	1	C	aa	161954	7	B	aa
153351	1	C	aa	162004	7	B	aa
153402	1	C	aa	162013	7	B	aa
153410	1	C	aa	162025	7	B	cc
153420	1	C	aa	162029	7	B	cc
153425	1	C	cc	162037	7	B	cc
153429	1	C	cc	162042	7	B	cc
153436	1	C	cc	162043	7	B	cc
153442	1	C	cc	162052	7	B	cc
153449	1	C	cc	162102	7	B	cc
153454	1	C	cc	162112	7	B	cc
153458	1	C	cc	162120	7	B	cc
153502	1	C	cc	162124	7	B	cc
153508	1	C	cc	162127	7	B	cc
153513	1	C	cc	162132	7	B	cc
153520	1	C	cc	162136	7	B	cc
153524	1	C	cc	162142	7	B	cc
153527	1	C	cc	162147	7	B	pc
153531	1	C	cc	162152	7	B	pc
153535	1	C	cc	162158	7	B	pc
153542	1	C	cc	162200	7	B	pc
153547	1	C	pc	162203	7	B	pc
153552	1	C	pc	162214	7	B	pc
153558	1	C	pc	162219	7	B	pc
153603	1	C	pc	162252	7	C	aa
153606	1	C	pc	162258	7	C	aa
153613	1	C	pc	162308	7	C	aa
153616	1	C	pc	162318	7	C	aa
153824	5	A	aa	162327	7	C	aa
153832	5	A	aa	162337	7	C	aa
153841	5	A	aa	162349	7	C	cc

Time	Point	Direction	Species	Time (cont.)	Point (cont.)	Direction (cont.)	Species (cont.)
153851	5	A	aa	162354	7	C	cc
153921	5	A	cc	162400	7	C	cc
153926	5	A	cc	162405	7	C	cc
153934	5	A	cc	162406	7	C	cc
153938	5	A	cc	162448	7	C	cc
153940	5	A	cc	162451	7	C	cc
153949	5	A	cc	162456	7	C	cc
153959	5	A	cc	162500	7	C	cc
154009	5	A	cc	162506	7	C	cc
154017	5	A	cc	162510	7	C	pc
154021	5	A	cc	162516	7	C	pc
154023	5	A	cc	162521	7	C	pc
154029	5	A	cc	162524	7	C	pc
154032	5	A	cc	162527	7	C	pc
154040	5	A	cc	162536	7	C	pc
154044	5	A	pc	162543	7	C	pc
154049	5	A	pc	162558	7	D	aa
154054	5	A	pc	162602	7	D	aa
154057	5	A	pc	162616	7	D	aa
154100	5	A	pc	162655	7	D	cc
154110	5	A	pc	162700	7	D	cc
154116	5	A	pc	162708	7	D	cc
154130	5	B	aa	162713	7	D	cc
154136	5	B	aa	162720	7	D	cc
154146	5	B	aa	162730	7	D	cc
154155	5	B	aa	162734	7	D	cc
154205	5	B	aa	162739	7	D	cc
154215	5	B	aa	162750	7	D	cc
154227	5	B	cc	162754	7	D	cc
154232	5	B	cc	162757	7	D	cc
154240	5	B	cc	162803	7	D	cc
154245	5	B	cc	162807	7	D	cc

Time	Point	Direction	Species	Time (cont.)	Point (cont.)	Direction (cont.)	Species (cont.)
154313	5	B	cc	162813	7	D	cc
154324	5	B	cc	162818	7	D	pc
154327	5	B	cc	162823	7	D	pc
154330	5	B	cc	162828	7	D	pc
154335	5	B	cc	162833	7	D	pc
154339	5	B	cc	162838	7	D	pc
154345	5	B	cc	162844	7	D	pc
154350	5	B	pc	162847	7	D	pc
154352	5	B	pc	162941	8	A	aa
154356	5	B	pc	162946	8	A	aa
154401	5	B	pc	162958	8	A	aa
154405	5	B	pc	163005	8	A	aa
154416	5	B	pc	163018	8	A	aa
154421	5	B	pc	163024	8	A	aa
154452	5	C	aa	163039	8	A	cc
154459	5	C	aa	163043	8	A	cc
154510	5	C	aa	163052	8	A	cc
154519	5	C	aa	163056	8	A	cc
154529	5	C	aa	163103	8	A	cc
154538	5	C	aa	163117	8	A	cc
154549	5	C	cc	163123	8	A	cc
154555	5	C	cc	163135	8	A	cc
154602	5	C	cc	163139	8	A	cc
154607	5	C	cc	163141	8	A	cc
154617	5	C	cc	163201	8	A	pc
154628	5	C	cc	163206	8	A	pc
154725	5	C	pc	163230	8	A	pc
154727	5	C	pc	163357	8	B	aa
154738	5	C	pc	163358	8	B	aa
154744	5	C	pc	163412	8	B	aa
154807	5	D	aa	163417	8	B	aa
154814	5	D	aa	163507	8	B	cc

Time	Point	Direction	Species	Time (cont.)	Point (cont.)	Direction (cont.)	Species (cont.)
154823	5	D	aa	163512	8	B	cc
154833	5	D	aa	163515	8	B	cc
154843	5	D	aa	163525	8	B	cc
154852	5	D	aa	163532	8	B	cc
154902	5	D	cc	163535	8	B	cc
154909	5	D	cc	163544	8	B	cc
154950	5	D	cc	163549	8	B	cc
155003	5	D	cc	163553	8	B	cc
155007	5	D	cc	163559	8	B	cc
155012	5	D	cc	163602	8	B	cc
155016	5	D	cc	163606	8	B	cc
155022	5	D	cc	163612	8	B	cc
155027	5	D	pc	163617	8	B	pc
155029	5	D	pc	163623	8	B	pc
155032	5	D	pc	163627	8	B	pc
155037	5	D	pc	163633	8	B	pc
155039	5	D	pc	163643	8	B	pc
155043	5	D	pc	163731	8	C	aa
155053	5	D	pc	163741	8	C	aa
155058	5	D	pc	163750	8	C	aa
155157	6	A	aa	163800	8	C	aa
155206	6	A	aa	163807	8	C	cc
155216	6	A	aa	163813	8	C	cc
155226	6	A	aa	163819	8	C	cc
155236	6	A	aa	163824	8	C	cc
155245	6	A	aa	163829	8	C	cc
155254	6	A	cc	163839	8	C	cc
155259	6	A	cc	163845	8	C	cc
155306	6	A	cc	163849	8	C	cc
155311	6	A	cc	163929	8	C	pc
155314	6	A	cc	163935	8	C	pc
155324	6	A	cc	163940	8	C	pc

Time	Point	Direction	Species	Time (cont.)	Point (cont.)	Direction (cont.)	Species (cont.)
155333	6	A	cc	163945	8	C	pc
155343	6	A	cc	163955	8	C	pc
155349	6	A	cc	164032	8	D	aa
155353	6	A	cc	164035	8	D	aa
155356	6	A	cc	164045	8	D	aa
155401	6	A	cc	164055	8	D	aa
155432	6	A	pc	164107	8	D	aa
155443	6	A	pc	164114	8	D	aa
155520	6	B	aa	164129	8	D	cc
155530	6	B	aa	164133	8	D	cc
155540	6	B	aa	164141	8	D	cc
155549	6	B	aa	164145	8	D	cc
155559	6	B	aa	164153	8	D	cc
155609	6	B	aa	164207	8	D	cc
155617	6	B	cc	164212	8	D	cc
155621	6	B	cc	164225	8	D	cc
155628	6	B	cc	164228	8	D	cc
155634	6	B	cc	164232	8	D	cc
155638	6	B	cc	164237	8	D	cc
155644	6	B	cc	164241	8	D	cc
155648	6	B	cc	164246	8	D	cc
155655	6	B	cc	164251	8	D	pc
155739	6	B	pc	164256	8	D	pc
155744	6	B	pc	164320	8	D	pc
155746	6	B	pc	—	—	—	—

A.11 Geodaten

Die erfassten und genutzten Geodaten (Sensor-, Signalpositionen, Lokalisierungen, Oberflächenmodell LIDAR Daten) sind über Zenodo veröffentlicht, siehe HOEDT (2018b).

A.12 Aufnahmedatensätze

Vollständigen Aufnahmen der Sensornetzwerks sind über Zenodo veröffentlicht, siehe HOEDT (2018c).