Acoular – Open-Source-Software zur Anwendung von Mikrofonarrayverfahren

Ennes Sarradj, Gert Herold

BTU Cottbus - Senftenberg, Lehrstuhl Technische Akustik
E-Mail: ennes.sarradj@b-tu.de, gert.herold@b-tu.de

Einleitung

Mikrofonarrayverfahren gehören heutzutage zu den Standardwerkzeugen in der Forschung und für Industrieanwendungen. Hauptmerkmal ist die zeitgleiche akustische Signalerfassung an mehreren Sensorpositionen und die anschließende Datenverarbeitung mit geeigneten Algorithmen. Ziel ist die Ermittelung der Position und Stärke gleichzeitig abstrahlender, räumlich verteilter Schallquellen.

Entsprechend ausgerüstete Software ist kommerziell erhältlich, allerdings ist hier eine Anpassung an nutzerspezifische Aufgaben und Randbedingungen oft nur begrenzt möglich. Zudem gestattet es eine Closed-Source-Architektur meist nicht, Änderungen oder Erweiterungen an den verwendeten Algorithmen vorzunehmen sowie deren genaue Arbeitsweise nachzuprüfen.

Eine Alternative stellt die quelloffene und frei verfügbare Software Acoular dar, die eine Reihe von Signalverarbeitungs- und Mikrofonarraymethoden implementiert [1]. Hierzu gehören unter anderem Beamformingverfahren im Frequenz- und Zeitbereich sowie diverse Entfaltungsverfahren. Die Betrachtung bewegter Quellen ist ebenso möglich wie die Kartierung räumlich verteilter Quellen in 3D. Die Schnittstellen sind so gestaltet, dass sich neue Algorithmen schnell einbinden lassen und der Programmablauf nachvollziehbar bleibt.

Dieser Beitrag umfasst eine Beschreibung der zugrunde liegenden Software-Architektur und einen typischen Signalverarbeitungsablauf. Einige Anwendungsbeispiele werden gezeigt und das Interface zur Implementierung neuer Algorithmen wird erläutert.

Problemstellung

Grundprinzip aller Mikrofonarrayverfahren ist, die Phasenunterschiede der zeitgleich an mehreren Positionen gemessenen Signale zu nutzen, um mithilfe eines Schallausbreitungsmodells Rückschlüsse auf den Ursprungsort der Schallquellen ziehen zu können.

Hierfür wird eine Anzahl von Fokuspunkten definiert, in deren Bereich Schallquellen vermutet werden. Aus den Abständen zwischen den Fokuspunkten (Index $n = 1 \dots N$) und Mikrofonen (Index $m = 1 \dots M$) lassen sich die Schalllaufzeiten Δt_{mn} zwischen diesen berechnen.

Die Idee des klassischen „Delay-and-Sum“-Beamformings im Zeitbereich ist, dass bei einer Aufsummierung zeitlich verschobener Mikrofonsignale immer dann ein Quellsignal hervortritt, wenn die Verschiebung gerade den Laufzeiten des Schalls von dieser Quelle zu den Mikrofonen entspricht:

$$p_n(t) = \sum_m h_{mn}\, p_m(t + \Delta t_{mn}) \; . \tag{1}$$

Der Koeffizient h_{mn} enthält dabei die Amplitudenkorrektur je nach Quellenmodell (Monopol, Dipol, etc.) sowie eine Gewichtung der Mikrofonsignale [2]. Um die Verteilung der Quellen zu kartieren, kann der quadratische zeitliche Mittelwert für jeden Fokuspunkt berechnet und als Pegel aufgetragen werden.

Für das Beamforming im Frequenzbereich wird zunächst die Kreuzspektralmatrix $\boldsymbol{C}$ nach der Methode von Welch aus den Zeitdaten errechnet [3]. Hierfür wird das Zeitsignal der Mikrofone in Blöcke geteilt, für jeden Block wird die FFT berechnet und die Kreuzspektren der Mikrofone werden über alle Blöcke gemittelt.

Die Schalldruckquadrate an einem Fokuspunkt für eine Frequenz lassen sich über

$$p_n{}^2 = \boldsymbol{h}_n^{\mathrm{H}}\, \boldsymbol{C}\, \boldsymbol{h}_n \tag{2}$$

berechnen, wobei der komplexwertige Steeringvektor $\boldsymbol{h}_n$ neben der Amplitudenkorrektur auch die Phasenverschiebung zwischen dem jeweiligen Fokuspunkt und den Mikrofonen enthält.

Zusätzlich zur Implementierung dieser grundlegenden Verfahren enthält das vorgestellte Programmpaket weitere Algorithmen zur Quellkartierung. Diese umfassen adaptive Verfahren, Verfahren, welche zusammenhängende Quellen identifizieren und separat kartieren können, Verfahren, die aufbauend auf dem klassischen Delay-and-Sum-Ergebnis die Ortsauflösung verbessern sowie inverse Verfahren [4–13].

Architektur

Acoular ist eine Python-Bibliothek und lässt sich als solche einfach in Python-Skripte einbinden und mit anderen Bibliotheken kombinieren. Das Python-Framework wurde bewusst gewählt, da sich hier die Erstellung von Skripten benutzerfreundlich gestaltet und eine Vielzahl an verfügbaren Bibliotheken eine flexible Arbeitsweise zur Datenverarbeitung und -darstellung erlaubt.

Rechenaufwändige Code-Bestandteile sind in C++ implementiert, womit durch zusätzliche Parallelisierung via OpenMP zeiteffiziente Berechnungen ermöglicht werden. Des Weiteren werden wichtige Zwischenergebnisse, wie etwa die berechnete Kreuzspektralmatrix als Ausgangspunkt für Beamforming im Frequenzbereich, durch einen intelligenten Caching-Mechanismus automatisch abgespeichert, um unnötige wiederholte Berechnungen zu vermeiden.

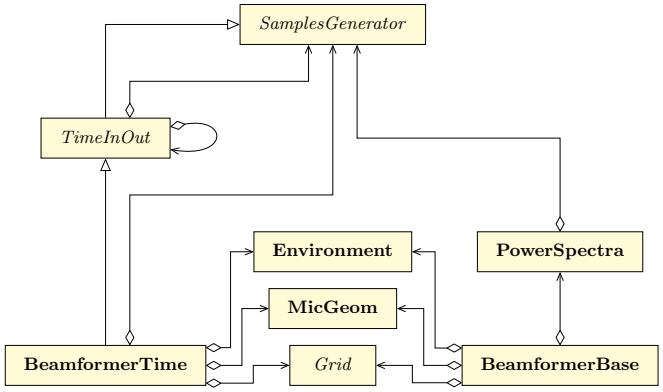

Abbildung 1: UML-Diagramm der Hauptklassen in Acoular.

Um die Einbindung neuer oder die Erweiterung bestehender Algorithmen so einfach wie möglich zu gestalten, wurde ein objektorientierter Programmieransatz verwendet. Die wichtigsten Hauptklassen sowie ihre Relationen zueinander sind in Abb. 1 dargestellt. Die kursiv gekennzeichneten „abstrakten“ Klassen stellen keine eigenen Algorithmen bereit, sondern definieren lediglich die benötigten Schnittstellen. Weitere Klassen werden je nach Funktionalität von den Hauptklassen abgeleitet und können über die gleichen Schnittstellen mit den anderen Klassen kommunizieren.

Die klare Strukturierung in Klassen erlaubt daher neben der Implementierung neuer Methoden unter geringem Aufwand einen variable blockweise Aneinanderreihung von Objekten zur Datenverarbeitung.

Anwendungsbeispiele

Drei Monopolschallquellen

Zur Demonstration der Arbeit mit Acoular soll ein Fall mit drei unterschiedlich stark abstrahlenden Schallquellen betrachtet werden. Die Zeitdaten wurden ebenfalls mit Acoular synthetisch erstellt. Vorgegeben wurden dabei neben der Art und Position der Quellen die Signale (jeweils unkorreliertes, weißes Rauschen) sowie deren Schalldruck-RMS-Werte in 1 m Abstand (1 Pa, 0,7 Pa bzw. 0,5 Pa). Zusätzlich wurde ein Array mit 64 Mikrofonen definiert, an denen die Schalldrücke ausgewertet werden. Das Mikrofonarray hat eine Apertur von 38 cm und befindet sich 30 cm über der Ebene mit den drei Quellen. Die Anordnung der Mikrofone ist in Abb. 3(a) dargestellt.

Abbildung 2 zeigt ein kurzes Python-Skript, in dem eine exemplarische Auswertung der erzeugten Zeitdaten durchgeführt wird. In Zeile 3 wird ein Objekt `ts` erzeugt, das die „Messdaten“ einliest. Die Geometrie des Mikrofonarrays wird über das Objekt `mg`, definiert in Zeile 4, eingelesen. Es folgt die Definition eines `PowerSpectra`-Objektes, das die Daten und Parameter für die Berechnung der Kreuzspektralmatrix zugewiesen bekommt (Zeilen 5 bis 7). In den Zeilen 8 bis 10 wird ein regelmäßiges Rechteck-Fokusgitter `rg` definiert, auf welchem die Auswertung stattfinden soll. Das Objekt für den verwendeten Beamforming-Algorithmus (in dem Fall das klassische

```
1  import acoular
2
3  ts = acoular.TimeSamples( name = 'three_sources.h5' )
4  mg = acoular.MicGeom( from_file = 'array_64.xml' )
5  ps = acoular.PowerSpectra( time_data = ts,
6                             block_size = 128,
7                             window = 'Hanning' )
8  rg = acoular.RectGrid( x_min = -0.2, x_max = 0.2,
9                         y_min = -0.2, y_max = 0.2,
10                        z = 0.3, increment = 0.01 )
11 bf = acoular.BeamformerBase( freq_data = ps,
12                               grid = rg, mpos = mg )
13 Lm = acoular.L_p( bf.synthetic( 8000, 3 ) )
```

Abbildung 2: Beispielskript zur Anwendung des klassischen Delay-and-Sum Beamformings im Frequenzbereich.

Delay-and-Sum-Beamforming im Frequenzbereich) wird in den Zeilen 11 bis 12 mit dem `PowerSpectra`-Objekt, dem definierten Rechteckgitter `rg` sowie der Mikrofongeometrie `mg` initialisiert. Ohne weitere Angaben geht der Algorithmus von einem ruhenden Medium und Monopolquellen aus.

Bis zu diesem Punkt wurden noch keine Berechnungen durchgeführt, sondern lediglich Objekte initialisiert. Die eigentliche Berechnung startet mit dem Aufruf der objekteigenen Routine `bf.synthetic(...)` in Zeile 13. Hier werden analog zu Gleichung (2) Schalldruckquadrate ermittelt und direkt mit der Funktion `L_p` in Schalldruckpegel umgerechnet. In Abb. 3(b) sind diese grafisch mit einer Dynamik von 15 dB dargestellt.

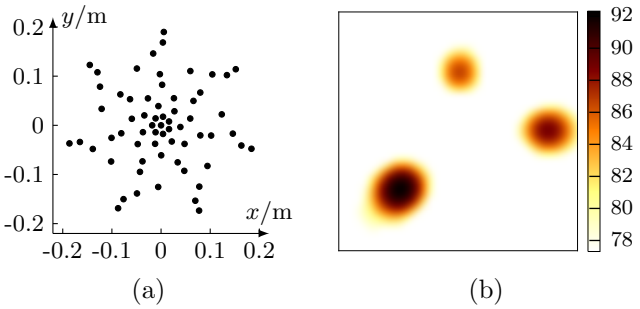

Abbildung 3: (a) Arraygeometrie mit 64 Mikrofonen. (b) Quellkarte mit drei Quellen für das Terzband um 8 kHz, Werte in dB.

Tragflügel im Freistrahl

Für dieses Beispiel werden experimentell gewonnene Daten mit mehreren Algorithmen ausgewertet. Der Messaufbau ist in Abb. 4 schematisch dargestellt: Hinter einer runden Düse ist ein Tragflügelprofil (Sehnenlänge 23,5 cm) befestigt, über das sowohl Kernstrahl als auch Scherschicht laufen.

Das gewählte Fokusgitter ist quadratisch mit einer Seitenlänge von 60 cm und liegt entlang der Sehne des Tragflügels. Es umfasst sowohl den Düsenaustritt als auch den Tragflügel sowie ca. eine Sehnenlänge Nachlauf. Das Mikrofonarray mit einer Apertur von 130 cm ist 68 cm darüber, parallel zur Fokusebene, angebracht.

Das ausgewertete Zeitsignal dauert 0,3 s und ist mit 51,2 kHz abgetastet. Für die Auswertungen im Frequenz-

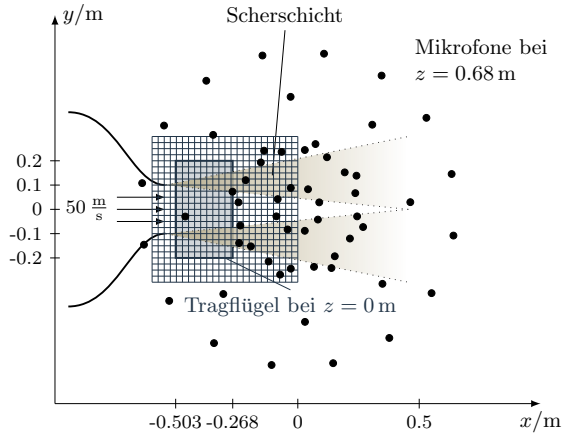

Abbildung 4: Tragflügelprofil im Freistrahl mit Mikrofonarray und quadratischem Fokusgitter (Draufsicht).

bereich ergeben sich bei einer Blockgröße von 128 Samples mit 50 % Überlappung damit 250 Kreuzspektren, aus deren Mittelung die Kreuzspektralmatrix approximiert wird.

In Abb. 5 sind die Quellkartierungen für das 4 kHz-Terzband, die sich aus der Anwendung 12 verschiedener Algorithmen auf den selben Datensatz ergeben, aufgetragen. Die Ergebnisse unterscheiden sich hinsichtlich der Ortsauflösung, des absoluten Pegels, des Dynamikumfangs sowie in Bezug auf die Art der rekonstruierten Quellen.

Je nach Verfahren sind Quellen am Düsenaustritt, aus der Interaktion der Scherschicht mit der Vorder- und Hinterkante sowie im Kernstrahl an der Tragflügelhinterkante zu erkennen. Zur Betrachtung der Zielstellung der jeweiligen Algorithmen und der zu erwartenden Ergebnisse sei auf die entsprechende Literatur verwiesen. Umfangreiche Untersuchungen an ähnlichen Konfigurationen, bei denen Acoular angewendet wurde, sind bereits in früheren Veröffentlichungen zu finden [12, 14–16].

Weitere Experimente, bei denen Acoular Anwendung fand, umfassen unter anderem die Betrachtung bewegter Quellen [17,18], die Kartierung von Quellen im dreidimensionalen Raum [15, 19], sowie Untersuchungen mit simulierten Quellen, um ein besseres Verständnis für die Eigenschaften der Verfahren und der Zuverlässigkeit der Modellannahmen zu erlangen [2, 20, 21].

Erweiterung

Exemplarisch für das Einbinden neuer Verfahren wird im Folgenden auf die Implementierung des „Functional Beamforming“ [13] in Acoular eingegangen. Das Verfahren arbeitet im Frequenzbereich, berechnet die Schalldrücke an den Fokuspunkten jedoch anstelle über Gl. (2) mit

$$p_n{}^2 = \left(\boldsymbol{h}_n^{\mathrm{H}} \boldsymbol{C}^{\frac{1}{\gamma}} \boldsymbol{h}_n \right)^{\gamma} \qquad (3)$$

und erreicht damit eine stärkere Unterdrückung der Nebenkeulenartefakte in den Quellkarten. Für die Berechnung von $\boldsymbol{C}^{\frac{1}{\gamma}}$ ist es notwendig, die Eigenwerte und -vektoren von $\boldsymbol{C}$ zu bestimmen.

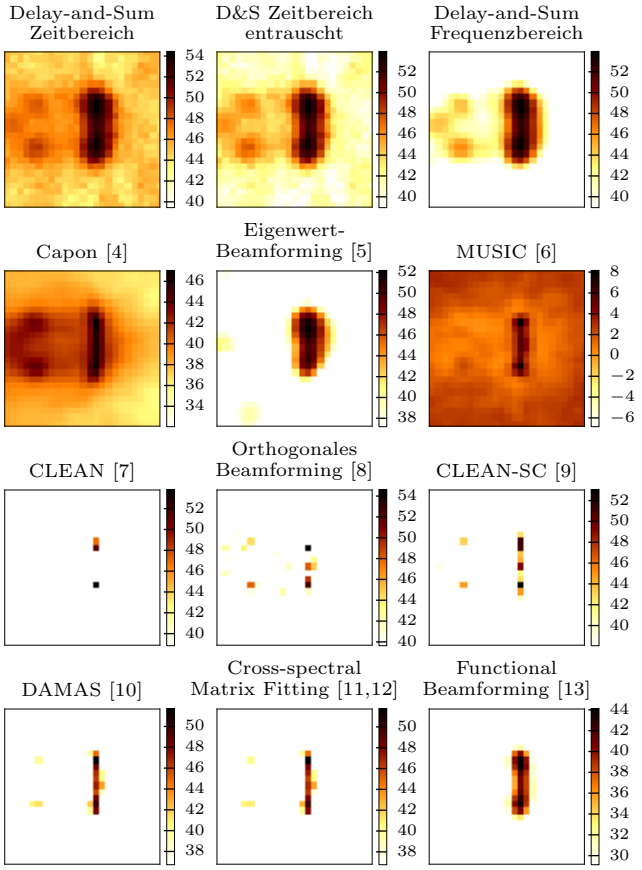

Abbildung 5: Mit unterschiedlichen Verfahren erstellte Quellkarten des umströmten Tragflügels (4 kHz-Terzband, Werte in dB).

Abbildung 6 zeigt den Quelltext für die Implementierung. Abgeleitet von der `BeamformerBase`-Hauptklasse wird eine neue Klasse `BeamformerFunctional` definiert. Diese erbt sämtliche Schnittstellen der Basisklasse. Nun müssen lediglich die abweichend von der `BeamformerBase`-Klasse benötigten Attribute und Funktionen definiert werden. Der Exponent γ bzw. `gamma` wird in Zeile 3 als neues reellwertiges Attribut mit dem Standardwert 1 definiert. Die zusätzliche Untereigenschaft `desc` ist bei der Einbindung der Klasse in eine grafische Oberfläche hilfreich. Das Attribut `freq_data` ist bereits in der Basisklasse vorhanden, allerdings dort durch die Klasseneigenschaften von `PowerSpectra` definiert. Durch die Initialisierung als `EigSpectra`-Objekt, welches neben der Kreuzspektralmatrix auch deren Eigenwerte und -vektoren übergeben kann (siehe Zeilen 17 und 18), werden die ursprünglichen Eigenschaften überschrieben.

Schließlich wird noch die Funktion `calc` neu definiert, in der eine Schleife alle benötigten Frequenzen durchläuft, für die Gl. (3) ausgewertet werden soll. Die eigentliche Berechnung wird durch eine in C++ implementierte Routine durchgeführt, welche bereits für andere Verfahren implementiert wurde und die in den Zeilen 21 und 22 aufgerufen wird.

```
class BeamformerFunctional( BeamformerBase ):

  gamma = Float(1, desc="functional exponent")

  freq_data = Trait(EigSpectra,
                    desc="freq data object")

  def calc(self, ac, fr):
    kj = 2j*pi*self.freq_data.fftfreq()/self.c
    numchannels = int(self.freq_data.numchannels)
    e = zeros((numchannels), 'D')
    h = empty((1, self.grid.size), 'd')
    beamfunc = self.get_beamfunc('_os')
    for i in self.freq_data.indices:
      if not fr[i]:
        eva = array(self.freq_data.eva[i][newaxis],
                    dtype='float64')**(1.0/self.gamma)
        eve = array(self.freq_data.eve[i][newaxis],
                    dtype='complex128')
        kji = kj[i, newaxis]
        beamfunc(e, h, self.r0, self.rm, kji,
                 eva, eve, 0, numchannels)
        ac[i] = h**self.gamma
        fr[i] = True
```

Abbildung 6: Implementierung des Functional Beamforming.

Zusammenfassung

Die freie und quelloffene Software Acoular zur Realisierung von Projekten rund um Mikrofonarrays wurde vorgestellt. Durch den offenen und multifunktionalen Ansatz ermöglicht das Programmpaket vielfältige Anwendungen.

Die modulare Architektur erlaubt den einfachen Aufbau von Datenverarbeitungsabläufen. Anhand zweier Anwendungsbeispiele wurden die Funktionsweise umrissen und einige implementierte Algorithmen demonstriert. Mithilfe der exemplarischen Implementierung eines neuen Mikrofonarrayverfahrens wurden zudem die Stärken des objektorientierten Programmaufbaus aufgezeigt.

Durch die Integrierung in die Python-Umgebung lässt sich Acoular flexibel mit anderen Modulen zur Datenverarbeitung und Visualisierung kombinieren.

Literatur

[1] Acoular – Acoustic testing and source mapping software, URL: `http://www.acoular.org/`, 03/2016

[2] Sarradj, E.: Three-Dimensional Acoustic Source Mapping with Different Beamforming Steering Vector Formulations. Advances in Acoustics and Vibration (2012), 1–12

[3] Welch, P. D.: The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms. IEEE Transactions on Audio and Electroacoustics (1967), 15(2), 70–73

[4] Capon, J.: High-resolution frequency-wavenumber spectrum analysis. Proceedings of the IEEE (1969), 57(8), 1408–1418

[5] Sarradj, E., Schulze, C., Zeibig, A.: Identification of noise source mechanisms using orthogonal beamforming. In Proceedings of International Congress on Noise and Vibration Emerging Methods (NOVEM 2005)

[6] Schmidt, R.: Multiple emitter location and signal parameter estimation. IEEE Transactions on Antennas and Propagation (1986), 34(3), 276–280

[7] Högbom, J. A.: Aperture synthesis with a non-regular distribution of interferometer baselines. Astronomy and Astrophysics Supplement (1974), 15(3), 417–426

[8] Sarradj, E.: A fast signal subspace approach for the determination of absolute levels from phased microphone array measurements. Journal of Sound and Vibration (2010), 329(9), 1553–1569

[9] Sijtsma, P.: CLEAN based on spatial source coherence. Int. Journal of Aeroacoustics (2007), 6(4), 357–374

[10] Brooks, T. F., Humphreys, W. M.: A deconvolution approach for the mapping of acoustic sources (DAMAS) determined from phased microphone arrays. J. Sound Vibration (2006), 294(4-5), 856–879

[11] Yardibi, T. et al.: Sparsity constrained deconvolution approaches for acoustic source mapping. J. Acoust. Soc. America (2008), 123(5), 2631–2642

[12] Herold, G., Sarradj, E., Geyer, T.: Covariance Matrix Fitting for Aeroacoustic Application. In AIA-DAGA 2013 Conference on Acoustics (pp. 1926–1928)

[13] Dougherty, R. P.: Functional beamforming. In Proceedings of the 5th Berlin Beamforming Conference 2014 (pp. 1–25)

[14] Geyer, T., Sarradj, E., Fritzsche, C.: Measurement of the noise generation at the trailing edge of porous airfoils. Experiments in Fluids (2010), 48(2), 291–308

[15] Geyer, T., Sarradj, E., Giesler, J.: Application of a beamforming technique to the measurement of airfoil leading edge noise. Advances in Acoustics and Vibration (2012)

[16] Geyer, T., Sarradj, E., Fritzsche, C.: Silent owl flight: Comparative acoustic wind tunnel measurements on prepared wings. Acta Acustica United with Acustica (2013), 99(1), 139–153

[17] Sarradj, E., Fritzsche, C., Geyer, T.: Silent Owl Flight: Bird Flyover Noise Measurements. AIAA Journal (2011), 49(4), 769–779

[18] Ballesteros, J. A. et al.: Noise source identification with Beamforming in the pass-by of a car. Applied Acoustics (2015), 93, 106–119

[19] Herold, G., Sarradj, E.: Mikrofonarrayverfahren zur Messung der abgestrahlten Schallleistung von Motorkomponenten. In Fortschritte der Akustik - DAGA 2015 (pp. 88–90)

[20] Herold, G., Sarradj, E.: Vergleich von Mikrofonarrayverfahren zur Schallquellencharakterisierung. In Fortschritte der Akustik - DAGA 2014 (pp. 624–625)

[21] Herold, G., Sarradj, E.: An Approach to Estimate the Reliability of Microphone Array Methods. In Proceedings of the 21st AIAA/CEAS Aeroacoustics Conference 2015 (pp. 1–10)