**DOI:** 10.23784/HN119-07

# Interactive processing of MBES bathymetry and backscatter data using Jupyter Notebook and Python

*An article by SOPHIE ANDREE*

This work explores the potential of Jupyter Notebook and Python to create an interactive processing tool for multibeam bathymetry and backscatter data. For this purpose, a Kongsberg EM 122 data set was used to identify and implement the required processing steps. Special attention was paid to the integration of freely available open source libraries to meet the performance requirements. The result is a modular approach that first decodes the required raw data, then computes the bathymetry and backscatter point clouds and finally applies semi-automatic filters to clean the bathymetry from outliers and visually correct the backscatter. Validation with data already processed on board confirmed the general feasibility of the approach. However, minor inconsistencies were encountered in the preprocessing of the bathymetry, which should be addressed in further work. Additionally, the tool can be extended for tidal correction and navigation processing.

Python | Jupyter Notebook | multibeam processing | bathymetry | backscatter | open source
Python | Jupyter Notebook | Verarbeitung von Fächerlotdaten | Bathymetrie | Backscatter | Open Source

Diese Arbeit untersucht das Potenzial von Jupyter Notebook und Python zur Erstellung eines interaktiven Tools zur Verarbeitung von Fächerlot-Bathymetrie- und -Rückstreudaten. Zu diesem Zweck wurde ein Kongsberg-EM-122-Datensatz verwendet, um die erforderlichen Verarbeitungsschritte zu identifizieren und zu implementieren. Besonderes Augenmerk wurde auf die Integration von frei verfügbaren Open-Source-Bibliotheken gelegt, um den Leistungsanforderungen gerecht zu werden. Das Ergebnis ist ein modularer Ansatz, der zunächst die benötigten Rohdaten decodiert, dann die Bathymetrie- und Rückstreupunktwolken berechnet und schließlich halbautomatische Filter anwendet, um die Bathymetrie von Ausreißern zu bereinigen und die Rückstreuung visuell zu korrigieren. Die Validierung mit bereits an Bord verarbeiteten Daten bestätigte die generelle Machbarkeit des Ansatzes. Allerdings traten kleinere Unstimmigkeiten bei der Vorverarbeitung der Bathymetrie auf, die in weiteren Arbeiten behoben werden sollten. Zusätzlich kann das Tool für die Gezeitenkorrektur und die Navigationsverarbeitung erweitert werden.

**Author**
Sophie Andree holds a M.Sc. degree in Geodesy with specialisation in Hydrography from HafenCity University in Hamburg.

sophie-andree@web.de

## Introduction

Today, multibeam echo sounders (MBES) are the most common and efficient method of conducting hydrographic surveys for the collection of bathymetry and backscatter data. Due to the challenging conditions in the marine environment and the complex MBES system setup, both types of data require various processing steps to provide reliable results. Conventionally, proprietary software suites with purchasable licenses are used for this purpose. While these leave little to be desired in terms of functionality and reliability, it is often semi-transparent what processing is applied to the data. In various areas of geomatics, there is a movement towards openness: keyword open source. Particularly in the university environment, open source and the use in teaching can be combined well in order to give students an understanding of the fundamental interrelationships. With this in mind, the idea was formed to investigate the possibilities of processing MBES data in order to develop an interactive tool based on Jupyter Notebook and Python. The exemplary data was acquired using a Kongsberg EM 122 MBES aboard the research vessel *Sonne* on a transit cruise (SO268-3) from Vancouver to Singapore, which is also available on PANGEA (Kinne et al., 2019).

## Why Python and Jupyter Notebook?

As a dynamically typed, interpreted programming language, Python requires relatively little code to express a high level of functionality. Therefore, the code is usually easy to read and quick to debug and review. When it comes to executing code, programming languages that are compiled in advance tend to be faster. However, especially for non-professional programmers, the readability and efficiency of code implementation in Python is often comparatively more productive than faster code execution. This is perhaps one of the reasons why Python has become particularly popular in the data science community (Carbonnelle 2020). This leads to another aspect that should be considered: The open source community for Python is large. Libraries and packages already exist for many different applications to solve a wide variety of tasks. In addition, Python can be used to extend and embed other languages such as C or C++. In this context, it can be understood as a »glue« language. Performance-critical parts of a program can be written in or adapted from faster languages, while Python is used for code control and adaptation (van Rossum and Drake 2003).

Jupyter Notebook is a free, open source, interactive web tool that is structured like a notebook. Software code, computational results, explanatory text and multimedia resources can be combined into a single document. Originally, the Jupyter project grew out of the IPython (interactive Python) project, with the goal of supporting interactive data science and scientific computing (Perkel 2018). In teaching, Jupyter Notebooks are particularly well suited for interactive software guides. IPython widgets (GUI controls) can be used to execute code specifically on user input without having to modify the actual code. In this way, a graphical user interface can be created with little effort.

## Input data

Kongsberg provides MBES data in the binary decoded ALL format. The individual sensor measurements (echo sounder, position system, motion sensor, etc.) are streamed to the output file as sequential datagrams. There are two different datagram types for both, bathymetry and backscatter data. For bathymetry, the choice is between the XYZ datagram and the raw range and angle datagram. The former includes the local Cartesian coordinates per beam computed in real time. Ship motion, sound velocity at the transducer face, and ray bending through the water column have already been corrected (Kongsberg 2018). The XYZ datagram was chosen for the first implementation, even though it disables corrections to the individual sensor data. The raw range and angle datagram could be integrated in a later stage.

For the backscatter, the choice was between the single value per beam reflectivity provided as part of the XYZ datagram or the beam time series written to the seabed image datagram. The single values are typically determined as some sort of average of the beam time series. This has the effect of discarding much of the original resolution. However, while the individual values are already georeferenced via the associated beam bathymetry, the individual beam time series samples must be georeferenced via the swath bathymetry in an additional processing step. Since the resolution of the backscatter data is very important, the seabed image datagram is used.

For georeferencing, the position datagram is needed as well (Fig. 1). The three identified datagrams can be decoded using the PyALL module written and published on GitHub by Kennedy (2016). Afterwards, the bathymetry is transformed from local Cartesian ship coordinates to global geographic coordinates using the navigation information. Then, the beam time series backscatter data are georeferenced by interpolating between the bathymetry soundings. Subsequently, the bathymetry and backscatter data are available as point clouds in a global coordinate reference frame. As a next step, a semi-automatic filtering of the bathymetry point cloud and visual image corrections for the backscatter data follow.

## Concept development

During concept development, a database approach was initially considered. As already discussed, the raw data are provided in different datagram types that have to be handled individually. However, the datagram timestamps are not always sequential because individual sensors sometimes output their measurements with a delay. For these types of tasks, a database-based approach offers an optimal solution. The basic idea is to store the data itself, the different processing stages, but also metadata and survey information across files in a project database. In this way, for example, a bathymetry sounding can be traced back through the processing to the original datagrams (XYZ and position). Likewise, it would be possible to query which soundings were measured in equidistant mode or to find all soundings between specific coordinates or dates.

However, during the database setup, it became clear that a complex database management was critical for both data integrity and processing performance. This phase proved to be extremely time consuming and there were only few exist-

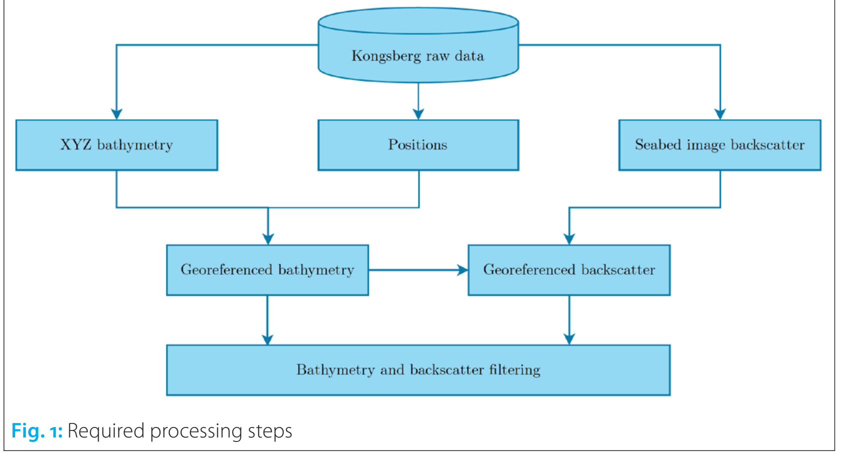


**Fig. 1:** Required processing steps

ing libraries that could have been effectively integrated. As a result, the approach failed in terms of performance. Firstly, performance in terms of the implementation itself, but also because the data volumes could not be handled without performance orientation.

The important lesson learned from the first approach was to move from an isolated monolithic approach based on a project database to a modular approach with less tightly coupled code. The individual modules must be specialised for processing large volumes of spatial data. In best case, the modules should come from established, open source libraries and may also come from performance-optimised programming languages such as C or C++ due to Python's ability to embed other languages.

With regard to the above criteria, a combination consisting of PDAL, Entwine and Potree was taken. PDAL (Point Data Abstraction Library) is a C++ based library for processing point clouds (PDAL Contributors 2020). The basic concept behind PDAL is the compilation of individual processing steps into pipelines. For example, spatial outlier filters can be assembled for bathymetry cleaning or attribute-based filters for backscatter correction. Entwine is a point cloud organisation software that uses an octree-based storage format (Hobu 2019). An octree is a tree data structure used for spatial indexing (Fig. 2). Individual points are indexed by incrementally dividing cuboids into eight child cuboids. Using a spatial index can speed up the processing of point clouds. The Entwine format can be read into PDAL pipelines and visualised in Potree. The latter is an interactive, WebGL-based point cloud renderer for large point clouds (Schütz 2020). It can be embedded in Jupyter Notebook, as it is also web-based and uses Entwine's Octree structure for efficient visualisation.

The compiled open source libraries are used to provide decoding of the identified datagrams from the raw Kongsberg data (PyALL) on the one hand,

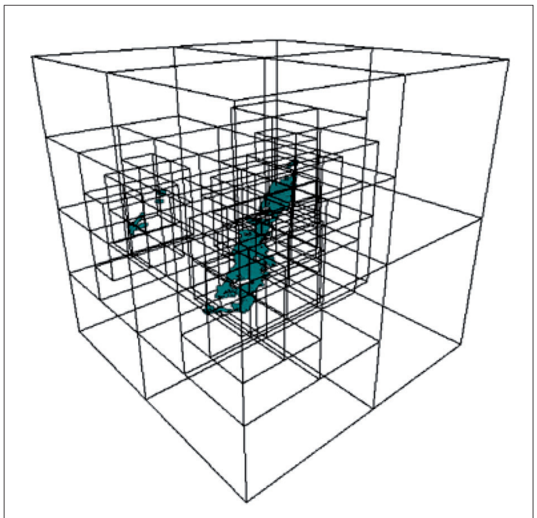
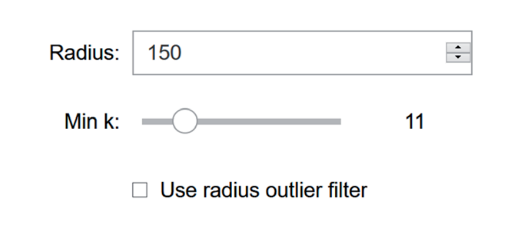


**Fig. 2:** Octree structure of example data created in Potree


Radius: 150

Min k: 11

☐ Use radius outlier filter

**Fig. 3:** GUI controls for the radius outlier filter

and bathymetry and backscatter point cloud filtering (PDAL, Entwine and Potree) on the other. To connect the two components, the raw data needs to be processed into point clouds by georeferencing. For this purpose, a separate Python module was written. The first step is to interpolate the ship positions to the ping timestamps. The heading is then used to transform the Cartesian coordinates of the individual soundings into the superior geographic coordinate system. For georeferencing the backscatter time series, the samples belonging to the bottom detection are identified. Since the georeferencing of the bottom detections is given by the bathymetry, the other samples can be georeferenced by rearranging the beam time series between the bottom detections and interpolating between the bottom detection samples.

As previously mentioned, no corrections are applied to the individual sensor data, which can significantly degrade the data quality. To address this problem, the processing was divided into two phases: First, a preprocessing Python module from raw Kongsberg data to point clouds. If any corrections would need to be applied, this step may be conducted within another software. The raw point clouds can then be imported as ASCII files and processed in a Jupyter Notebook in which they are further filtered to outlier-cleaned bathymetry and visually corrected backscatter. Thereby the filters can be configured via GUI controls. For the bathymetry outlier cleaning, a combination of a depth window filter, an extended local minimum (ELM) filter, a radius (Fig. 3) and a statistical outlier filter can be used. For the backscatter corrections, a constant offset or multiplier can be applied, a median absolute deviation (MAD) filter used for despeckling and Poisson sampling for anti-aliasing.

## Results

To act in the spirit of open source, the tool has been released on the code hosting platform GitHub under the MIT license (Andree 2021). There is also a documentation/manual in the repository that provides a good overview if there is interest in the code itself and how to use it. For now, only the results of the bathymetry and backscatter processing are considered here. For evaluation, the grids created during the cruise from manually cleaned soundings and backscatter mosaics are taken.

The result of the bathymetry processing (Fig. 4)

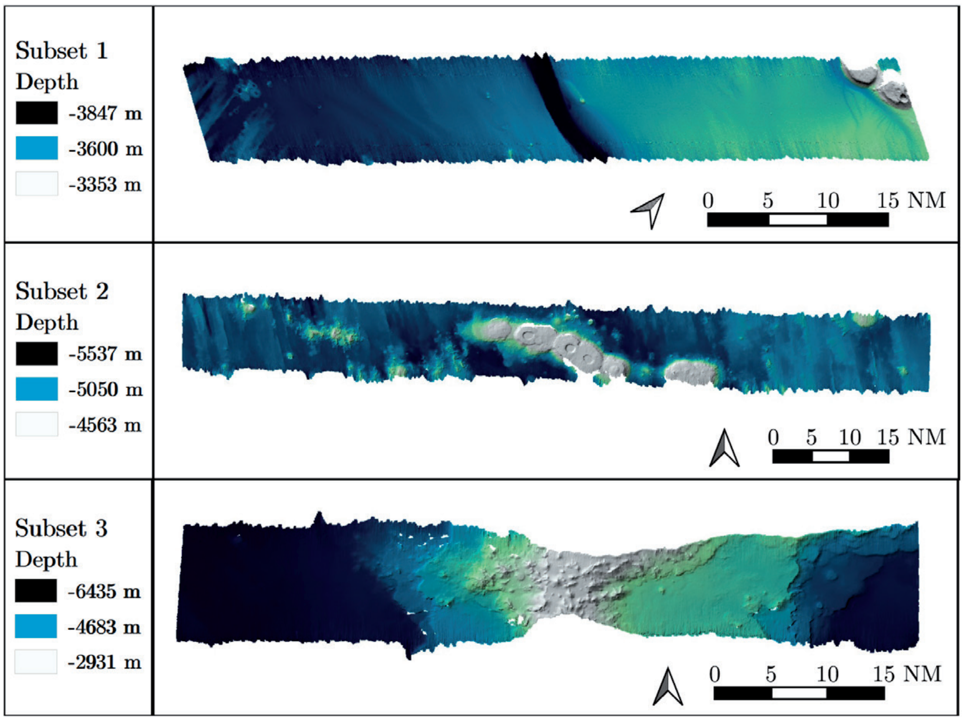


**Fig. 4:** Processed bathymetry

looks consistent. Most of the outliers could be filtered well except for areas where they accumulate. When compared to the on-board processed bathymetry (Fig. 5 and Fig. 6), two patterns were noted: There are clearly recognisable offsets around bathymetric features and in the areas of the outer beams.

The pattern is seen in all analysed data subsets and appears to be related to the travel direction of the ship. At this point, it is not clear at which step

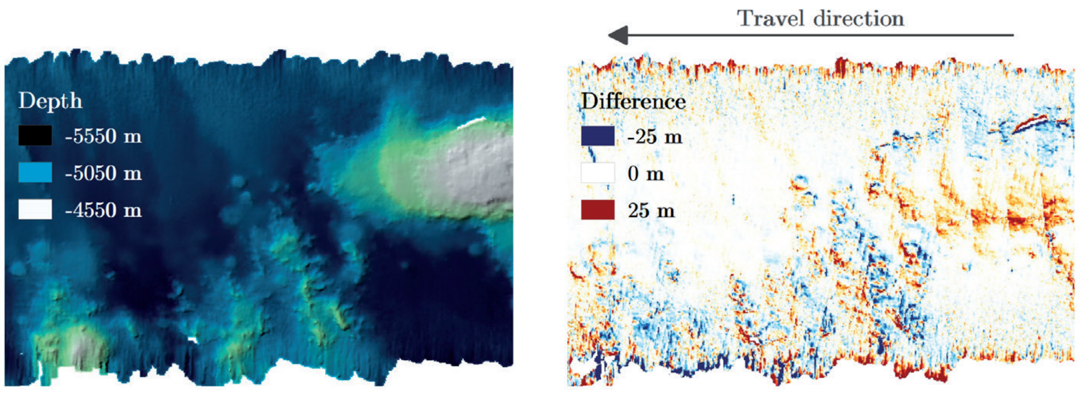


**Fig. 5:** Left: Section of processed subset 2 bathymetry. Right: Difference to reference bathymetry

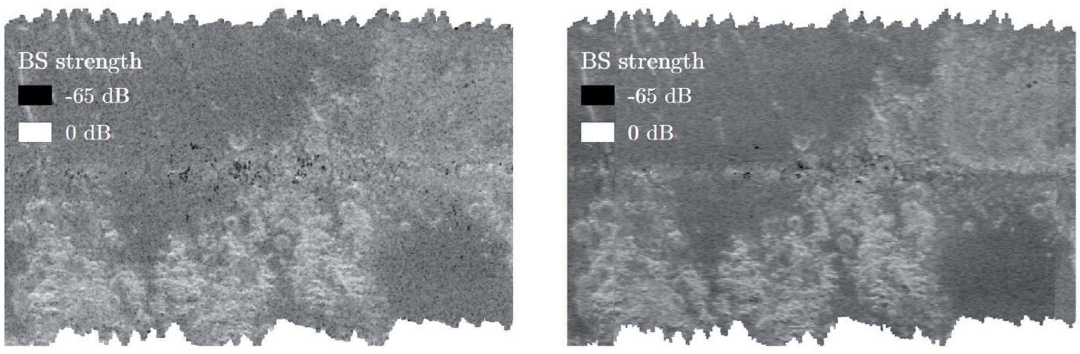


**Fig. 6:** Left: Section of processed subset 2 backscatter. Right: Reference backscatter

this deviation occurs. Presumably, it is due to a difference in preprocessing. Based on the correlation with the direction of travel, the cause can probably be narrowed down to a time or pitch offset. A possible pitch offset could be caused by a difference in the handling of the dual-swath mode. A time offset could result from a discrepancy in the definition of the exact measurement time or the position interpolation itself.

The georeferencing of the beam time series looks plausible and correctly represents bathymetric features. Since only visual and no comprehensive radiometric corrections were applied, strictly speaking the two data sets cannot be compared. Nevertheless, the visual corrections served their purpose (Fig. 6). In addition, the Kongsberg real-time corrections were found to give very pleasing results. Compared to the reference grid, it can be seen that the overall backscatter range is slightly different and that the result is more noisy and speckled with some remaining artefacts around the centre beam area.

## Conclusion and outlook

The idea of this work was to develop a tool for processing bathymetry and backscatter data from Kongsberg EM series MBESs. This could be realised using freely available open source libraries. The advantages of the concept are the modular approach especially in terms of long-term maintainability and adaptability. The entire tool is purely open source and can therefore be used by anyone. By integrating C++ libraries, the performance requirements could be met.

The disadvantages of this approach are that interactivity comes at a high price in terms of complexity. GUI programming must be well thought out to avoid becoming clumsy. Also, Jupyter Notebook is not designed for pure GUI programming, as the code execution is very flexible. It is practical and fast for small programs but cannot be scaled up arbitrarily. Potentially, the dependence on existing libraries can also be disadvantageous since the functionality is so externally determined.

Overall, the use in teaching is advantageous in that the students work much closer to the raw data and have a closer contact to programming. In this context, it may also be useful to reduce the GUI programming. The tool that was developed in the course of this work can be understood as a rough framework, which can be optimised in the following. Important successive steps would be the identification and elimination of the position offset in the bathymetry. In addition, further filters for both bathymetry and backscatter can be added very easily. More profound improvements could be the integration of tide correction possibilities and navigation processing. //

**References**

Andree, Sophie (2021): Interactive MBES processing. https://github.com/SophieHCU/Interactive-MBES-processing

Carbonnelle, Pierre (2020): PYPL. PopularitY of Programming Language index. https://pypl.github.io/PYPL.html

Hobu, Inc. (2019): Entwine. Version 2.1. https://github.com/connormanning/entwine

Kennedy, Paul (2016): PyALL. Version 1.50. https://github.com/pktrigg/pyall

Kinne, Stefan; Annika Jahnke et al. (2019): MICRO-FATE – Characterization of the fate and effects of microplastic particles between hotspots and remote regions in the Pacific Ocean, MORE-2 – Measuring Ocean REferences (of aerosol, clouds and trace-gases for evaluations of satellite retrievals and model simulations) – part 2. SONNE-Berichte, Cruise SO268-3, 30.05.2019–05.07.2019, Vancouver–Singapore

Kongsberg (2018): EM Series. Multibeam echo sounders. Datagram formats

PDAL Contributors (2020): PDAL. Point Data Abstraction Library. Version 2.2.0. https://github.com/PDAL/PDAL

Perkel, Jeffrey M. (2018): Why Jupyter is data scientists' computational notebook of choice. Nature, DOI: 10.1038/d41586-018-07196-1

Schütz, Markus (2020): Potree. Version 1.7. https://github.com/potree/potree

van Rossum, Guido; Fred L. Drake (2003): An introduction to Python. Release 2.2.2. Network Theory Ltd.