SoftwareX 34 (2026) 102670

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

py4dgeo: Open-source scientific software for topographic change analysis in 3D/4D geographic point clouds

K. Anders [a,*], D. Kempf [b], W. Albert [c], P. Andriushchenko [b,d], X. Huang [a], D. Hulskemper [e], T. Isensee [b], D. Kapitan [b], R. Tabernig [c], H. Weiser [c,f], L. Winiwarter [g], V. Zahs [c,h], B. Höfle [c,f]

[a] Remote Sensing Applications, TUM School of Engineering and Design, Technical University of Munich (TUM), Ottobrunn, Germany
[b] Scientific Software Center (SSC), Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Heidelberg, Germany
[c] 3DGeo Research Group, Institute of Geography, Heidelberg University, Heidelberg, Germany
[d] Heidelberg Institute of Global Health (HIGH), Heidelberg University Hospital (UKHD), Heidelberg University, Heidelberg, Germany
[e] Department of Geoscience & Remote Sensing, Delft University of Technology, Delft, the Netherlands
[f] Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Heidelberg, Germany
[g] Unit of Geometry and Surveying, Faculty of Engineering Sciences, University of Innsbruck, Innsbruck, Austria
[h] RBS wave GmbH, Ettlingen, Germany

HIGHLIGHTS

- Automated and reproducible workflows for 3D time series analysis.
- Comprehensive processing of massive point cloud data across the full change analysis pipeline.
- Easy coupling and exchange of state-of-the-art change analysis methods.
- Flexible interfaces to common geospatial software and scientific Python tools.
- Multi-institutional development board including research software engineers.

ARTICLE INFO

Keywords:
Python library
Change detection
Spatiotemporal analysis
Topographic time series
Geospatial software
M3C2 algorithm
Point cloud registration

ABSTRACT

The software py4dgeo is an open-source Python library for automated analysis of 3D/4D geographic point clouds with a focus on topographic change detection and quantification, and on surface dynamics analysis. py4dgeo addresses a growing need for robust, reproducible, and extensible tools capable of handling complex spatiotemporal datasets, especially for geographic applications and topographic monitoring in general. The library implements state-of-the-art methods for point cloud registration, bitemporal 3D change analysis, and time series-based approaches. py4dgeo features a modular architecture combining a C++ core designed for computationally demanding tasks with a flexible Python interface, enabling robust processing of large datasets while supporting rapid scientific method development. Its object-oriented data structures manage temporal point cloud series, core points, and spatiotemporal neighborhoods in a transparent and extensible way. Full interoperability with widely used tools such as PDAL, CloudCompare, and the Python ecosystem (e.g., via LAS/LAZ format and NumPy arrays) facilitates seamless integration into existing workflows. The library is accompanied by comprehensive documentation, open data-based Jupyter demos, and community-driven development to support reproducibility and encourage contributions. py4dgeo provides a scalable foundation for monitoring dynamic 3D environments and is already used in numerous research and application projects for automated, quantitative change analysis in 4D point clouds. Therefore, py4dgeo contributes an essential resource to the growing field of spatiotemporal analysis of geographic point cloud data.

* Corresponding author.
Email address: k.anders@tum.de (K. Anders).

https://doi.org/10.1016/j.softx.2026.102670
Received 9 February 2026; Received in revised form 30 March 2026; Accepted 18 April 2026
Available online 25 April 2026

Metadata

Code metadata (mandatory).

Nr.	Code metadata description	Metadata
C1	Current code version	v1.1.0
C2	Permanent link to code/repository used for this code version	https://github.com/3dgeo-heidelberg/py4dgeo
C3	Permanent link to Reproducible Capsule	None
C4	Legal Code License	MIT License
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python, C++, Jupyter Notebook, CMake, scikit-build-core, Eigen, nanoflann, GitHub Actions
C7	Compilation requirements, operating environments & dependencies	Operating on Linux, Windows, and macOS with 64-bit Python ≥ 3.9. Building from source requires a C++17-compliant compiler and optionally Doxygen (documentation building is skipped if missing).
C8	If available Link to developer documentation/manual	https://py4dgeo.readthedocs.io
C9	Support email for questions	Please use the issue tracker of the GitHub repository.

1. Motivation and significance

The processing and interpretation of multitemporal geographic point clouds, that is three-dimensional (3D) geospatial data acquired repeatedly over time, has gained increasing relevance due to the growing availability of high-resolution topographic data from terrestrial, airborne, and mobile laser scanning, as well as photogrammetric reconstruction [1–3]. Applications span diverse domains such as environmental monitoring, natural hazard assessment, infrastructure inspection, and autonomous mapping [e.g.,4–8]. These sensing strategies enable observations at temporal intervals ranging from seconds to seasons, resulting in dense time series of 3D scenes. Massive spatiotemporal data are generated especially when data acquisition is frequently repeated at near-continuous intervals with respect to the observed process and over long observation periods [2,9]. The resulting 4D (3D + time) point clouds require dedicated analysis methods to detect and quantify change. Established point cloud processing frameworks, such as PDAL, PCL, LAStools, and CloudCompare [10], provide extensive capabilities for geometric filtering, visualization, and spatial data management. For change analysis workflows, they offer algorithms for typical bitemporal operations, such as co-registration or cloud-to-cloud distance computation. CloudCompare and PDAL both provide an implementation of the established Multiscale Model to Model Cloud Comparison (M3C2) algorithm [11]. In CloudCompare, the Python wrapper CloudComPy provides a scriptable interface to the methods available in the graphical software [12]. A Python framework dedicated to managing point cloud time series is provided by pointcloudset [13], but without specific processing methodology. The remaining gap lies in the lack of a scriptable, method-oriented framework that integrates established and emerging multitemporal change analysis techniques into a coherent, single-interface workflow, enabling automated, reproducible and multi-method analysis of multitemporal 3D and 4D geographic point clouds. The key concept of py4dgeo is to bundle different components of the change analysis pipeline in a modular way, thereby providing easy access for users to integrate into different workflows and objectives (Fig. 1). In summary, py4dgeo provides a scientific software platform focused specifically on multitemporal point cloud analysis, bridging the gap between general-purpose point cloud tools and the specialized needs of spatiotemporal change detection in multitemporal 3D and 4D geospatial data.

2. Software description

As an open-source Python library for the analysis of change in 3D/4D point clouds, py4dgeo is developed with a hybrid architecture that combines the usability of Python with the performance of C++. The Python interface provides user-friendly access to all functionalities and enables integration with the broader Python scientific ecosystem. The library is distributed under the MIT license and hosted on GitHub

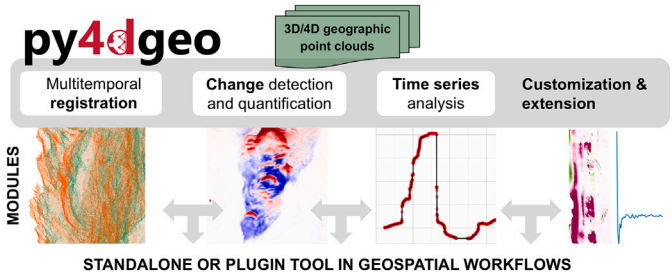

Fig. 1. Overview of the py4dgeo Python library, a modular tool that can be used standalone for 3D/4D geographic point cloud analysis or integrated into comprehensive workflows among other common open-source tools for geospatial data processing and visualization.

(https://github.com/3dgeo-heidelberg/py4dgeo). It is distributed both as a source distribution and as a pre-compiled binary distribution for Windows, macOS and Linux via the Python Package Index (PyPI, https://pypi.org/project/py4dgeo/). The comprehensive documentation includes installation guidelines, API references, and illustrative examples, and is automatically deployed to ReadTheDocs.org (https://py4dgeo.readthedocs.io). A suite of more advanced demos with real-world applications is provided through a gallery of self-contained Jupyter notebooks that use open datasets and support reproducibility by linking to peer-reviewed publications. py4dgeo follows an open development model that encourages community contributions. Software management and maintenance are steered by a multi-institution development board to ensure sustainability in terms of quality and consistency. This includes support from professional research software engineers, working in close collaboration with domain experts in geographic point cloud analysis. The design of interfaces and prioritization of new features are guided by feedback from users and contributors mainly through the issue tracker. Community contributions are reviewed via a public merge request process.

2.1. Software architecture

py4dgeo follows a layered architecture with a C++ backend, Python bindings via pybind11, and high-level Python interfaces. This design places computationally intensive components into the C++ core while the Python side provides flexibility for method development, visualization, and workflow composition. At the Python level, py4dgeo exposes a set of high-level classes and methods designed for scientific users. All core classes follow consistent interfaces, enabling users to substitute or chain different methods with minimal changes to their code. The library is fully compatible with scientific Python libraries such as

NumPy, scikit-learn, and (geo)pandas, allowing integration into custom data processing workflows. py4dgeo executes selected computational routines in parallel using CPU multi-threading where applicable. The framework currently provides CPU-based processing only and does not include a GPU backend. py4dgeo includes utilities for reading and writing the common point cloud formats LAS/LAZ and XYZ (that is, tabular plain text in general) via laspy and NumPy arrays. While py4dgeo does not include direct, built-in interfaces to external software such as PDAL or pointcloudset, it can be combined with these tools through shared data formats (e.g., LAS/LAZ, NumPy), enabling users to incorporate py4dgeo into broader geospatial or time series processing pipelines. Thereby, architectural design is guided mainly by principles of modularity, extensibility, interoperability, and reproducibility. Configuration of workflows is exposed explicitly, and demos are provided using open datasets and published methods. This promotes sustainable community development and lowers the barrier for implementing and publishing reproducible workflows in point cloud change analysis, making py4dgeo suitable for both exploratory and automated analysis in scientific and operational settings.

2.2. Software functionalities

py4dgeo implements a suite of core methods for analysis of 3D/4D point cloud data. These cover the entire typical change analysis pipeline including registration, change detection and quantification, and time series analysis, which are detailed in the following subsections. py4dgeo structures multitemporal point cloud data using a compact set of extensible abstractions (Fig. 2). These support different acquisition modes and sampling schemes while enabling generalized change detection workflows. The core unit of temporal data organization is the Epoch class, representing a single acquisition of a point cloud at a defined point in time. Each Epoch object stores the point cloud coordinates, optionally a timestamp, and further optional metadata (e.g., sensor model, acquisition method, registration accuracy) or precomputed attributes, such as expensive search tree data structures. For time series analysis, the SpatiotemporalAnalysis class encapsulates the entire analysis context for a list of Epoch instances together with a parametrized M3C2 algorithm, including the property definition of a reference epoch and of core points, i.e., 3D points at whose locations change is quantified. This can be an original epoch, a subset thereof, or separately defined coordinate locations, as developed in the original M3C2 algorithm [11] (Section 2.2.2). The abstraction allows time series-based change analysis to operate on a consistent interface, for example to assess temporal behavior at a core point location. The SpatiotemporalAnalysis object stores analysis results (e.g., change values, uncertainty variables, 4D objects).

2.2.1. Iterative closest point (ICP) registration with stable area estimation

Accurate multitemporal alignment of point clouds is essential for meaningful change analysis. py4dgeo includes a standard Iterative Closest Point (ICP) method [14], requiring two epochs and an optional reduction point that shifts coordinates for accurate rotation handling as inputs. Application of the returned transformation object (consisting of a 4×4 transformation matrix and a reduction point) to all points in the Epoch object is explicitly controlled by the user and recorded in the Epoch object. An enhanced ICP variant by Yang and Schwieger [15] supports automatic stable area estimation. This approach enables to scale the automated alignment across large 3D time series with minimal manual input, for example in challenging natural environments where fixed control points are unavailable. Practical requirements and potential failure modes of registration, as well as the subsequent change analysis methods, depend strongly on dataset characteristics and the specific scientific application. Users are therefore encouraged to consult the referenced method publications (cf. mainly Section 2.2) for detailed discussions of assumptions, validation steps, and recommended usage conditions and parametrization.

2.2.2. Surface change analysis with the M3C2 algorithm

py4dgeo implements the M3C2 algorithm [11] for surface change quantification. M3C2 computes signed distances between local neighborhoods along local surface normal vectors, with support for

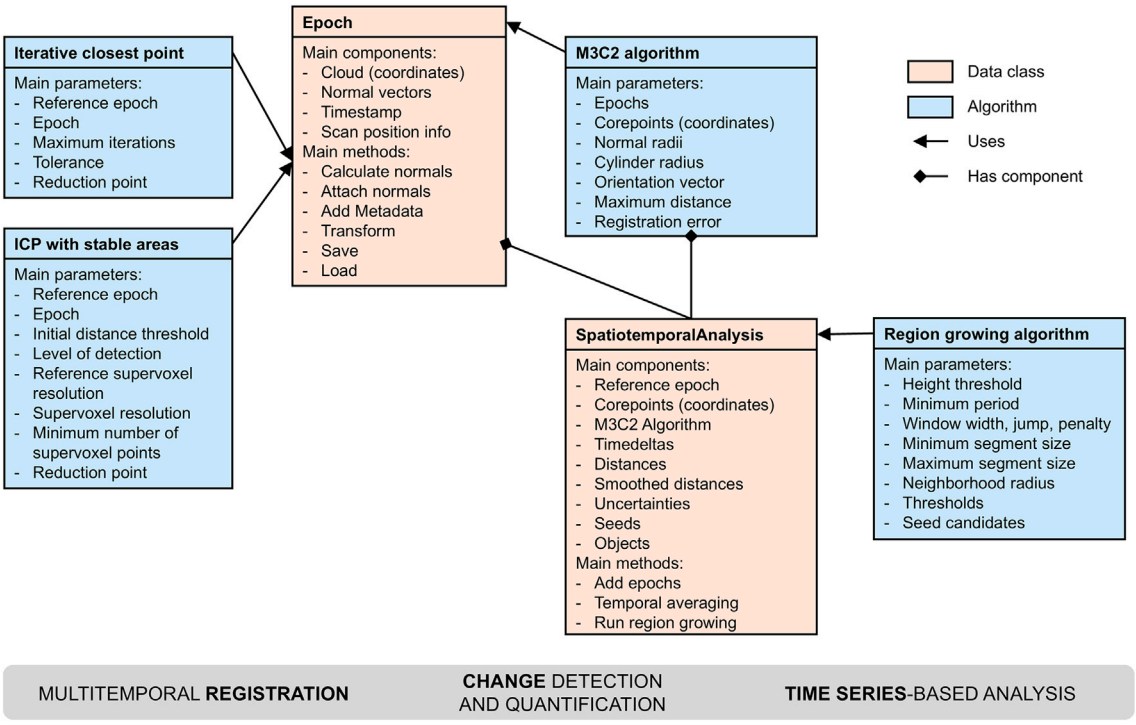

Fig. 2. Overview of main data and algorithm structures in py4dgeo representing the core modules of multitemporal registration, change detection and quantification, and time series-based analysis.

multi-scale normal estimation and uncertainty quantification through the level of detection. Implemented state-of-the-art variants of the M3C2 algorithm include the M3C2 with error propagation (M3C2-EP) by Winiwarter et al. [16] and the correspondence-driven plane-based M3C2 (PBM3C2) developed by Zahs et al. [17]. Runtime mainly scales with the number of core points and the cost of neighborhood searches (e.g., cylinder radius and maximum distance), while memory usage is driven by the number of points provided as input.

2.2.3. Time series-based analysis using the 4D objects-by-change algorithm

For dense 3D time series, py4dgeo provides 4D objects-by-change (4D-OBCs) extraction [18], extending standard bitemporal change analysis to consider the full temporal domain. Relevant surface activities are first detected in the time series of all core points as defined by the seed detection algorithm. Objects are spatially delineated based on their surface behavior over time using region growing segmentation with similarity between time series as homogeneity criterion. The resulting objects are stored in the SpatiotemporalAnalysis object with their spatial and temporal extents in terms of core point indices belonging to the object, and the index of start and end epoch in the time series. As part of typical 3D time series processing steps, py4dgeo further offers a time series filtering method that applies a rolling median within a user-defined temporal window to all time series in a SpatiotemporalAnalysis object. For this algorithm, runtime increases with the number of seeds and with the number of epochs (variable among seeds). The computational cost of Dynamic Time Warping (DTW)-based time series similarity grows approximately quadratically with the number of epochs.

2.3. Customization possibilities

Alongside its core methods, py4dgeo provides configurable access to internal algorithmic components, supporting both the adaptation of the software to specific scientific applications and low-barrier testing of new methodological developments. One example is the seed detection step in the 4D-OBC framework, which can be adapted using application-specific heuristics, such as the linear change behavior targeted by Ulm et al. [19]. Another example is the overriding of normal vector calculation by providing a fixed surface orientation or pre-computed normal vectors in the direction algorithm (cf. Listing 2, lines 22–25). py4dgeo provides documented interfaces and examples for such customizations. Customizations are implemented by providing callback functions for specific tasks. Callbacks can be written in Python or C++, allowing both rapid prototyping and performant implementation of established algorithms within the same framework. Built-in callbacks are implemented in both languages and used as regression tests.

2.4. Extension possibilities

The data structures and methods of py4dgeo can be easily used for extended analysis with the functionality of other tools or Python libraries, such as scikit-learn. For example, the SpatiotemporalAnalysis object provides the basis for k-means clustering to perform time series clustering following [20], see Section 3.2. Further extension examples are demonstrated by Frantzen [21] using principal component analysis to explore spatiotemporal patterns in the dataset, and by Hulskemper et al. [22] using extracted 4D objects-by-change as input for object grouping with self-organizing maps.

2.5. Best practices of research software engineering

py4dgeo is developed following best practices of research software engineering. Both the Python and C++ code are unit-tested using modern testing frameworks (pytest, Catch2). Execution of these tests on all target platforms (Windows, macOS, Linux) is integrated into the development workflow via GitHub Actions. The project employs continuous integration and code-coverage monitoring to ensure the robustness and sustainability of the code base. Our approach of using Jupyter notebooks to both render documentation and provide integration tests has been mentioned as an example of a successful approach to writing maintainable documentation by Fritz et al. [23]. py4dgeo relies on well-established performance-oriented C++ libraries like Eigen by Guennebaud et al. [24] and nanoflann by Blanco and Rai [25] to support computationally intensive components of the software. The project's open license is complemented by an open contribution workflow on GitHub.

3. Illustrative examples

To demonstrate the capabilities and typical use cases of py4dgeo, we present two illustrative examples. These examples reflect real-world scenarios of 3D/4D point cloud analysis covering key components of the library. The full examples are available as Jupyter notebooks among the demos in the software repository and are featured in the open-source e-learning course E-TRAINEE [26]. We use the publicly available point clouds provided within module 3 of the E-TRAINEE course via the Zenodo data repository [27].

3.1. Registration and bitemporal change analysis in complex topography

In the first example, our objective is to detect and quantify bitemporal surface change in the complex topography of an active rock glacier, the Äußeres Hochebenkar (AHK) in Tyrol, Austria [28]. The two LAS files from 2020 and 2021 are loaded into py4dgeo as Epoch objects. Then, the core points are defined to use a point cloud subset defined as every 100th point in the first epoch. Next, the M3C2 algorithm is parametrized to use multi-scale normal radii from 4.0 m to 6.0 m at 0.5 m intervals, and a cylinder radius of 0.5 m. The maximum search depth is set to 15.0 m, and the registration error is given by a determined alignment accuracy of 3.7 cm. The code workflow is depicted in Listing 1. The objects returned from the M3C2 algorithm contain the M3C2 distances at each core point and the associated uncertainties.

The results can be plotted with standard Python functionality, visualizing different layers of change information, such as the M3C2 distances, the directionality of the surface change (here, Z component of surface normal vectors), the level of detection, or areas of significant change where the distance value is greater than the level of detection value (Fig. 3). To use the results in separate software, the outputs can be stored as a point cloud file with all needed attributes attached to the core point coordinates. Layers can be rasterized for GIS processing or 2D map visualizations, for example, using PDAL directly in the Python script.

With additionally available metadata of the LiDAR sensor and survey properties, the advanced M3C2 variant with error propagation (M3C2-EP) following [16] can be applied. This requires the additional specification of scan position information and full co-registration information in the form of a covariance matrix, together with an affine transformation matrix and a reduction point defining the rotation origin. Each epoch must include a per-point scan position ID in its additional dimensions, linking every point to a scan position, which is required by the implementation for uncertainty propagation. Provided these additional parameters, the script workflow is similar to the standard M3C2. Corresponding to the derived distances, 3D covariance information for the point cloud is returned, and the estimation of the associated uncertainty is improved through error propagation from the acquisition parameters to the final level of detection estimation. The full example of M3C2-EP is given in a dedicated basic usage tutorial in py4dgeo.

```python
# load epochs into py4dgeo objects
epoch2020 = py4dgeo.read_from_las(pc_file_2020)
epoch2021 = py4dgeo.read_from_las(pc_file_2021)

# use every nth point as core point for distance calculation
corepoints = epoch2020.cloud[::100]

# instantiate and parametrize the M3C2 algorithm object
m3c2 = py4dgeo.M3C2(
    epochs=(epoch2020, epoch2021),
    corepoints=corepoints,
    normal_radii=[4.0, 4.5, 5.0, 5.5, 6.0]
    cyl_radius=0.5,
    max_distance=(15.0),
    registration_error=(0.037)
)

# run the M3C2 analysis
m3c2_distances, uncertainties = m3c2.run()
```

Listing 1. Code workflow to setup and run the M3C2 change quantification in py4dgeo for two point cloud epochs loaded from LAS files.

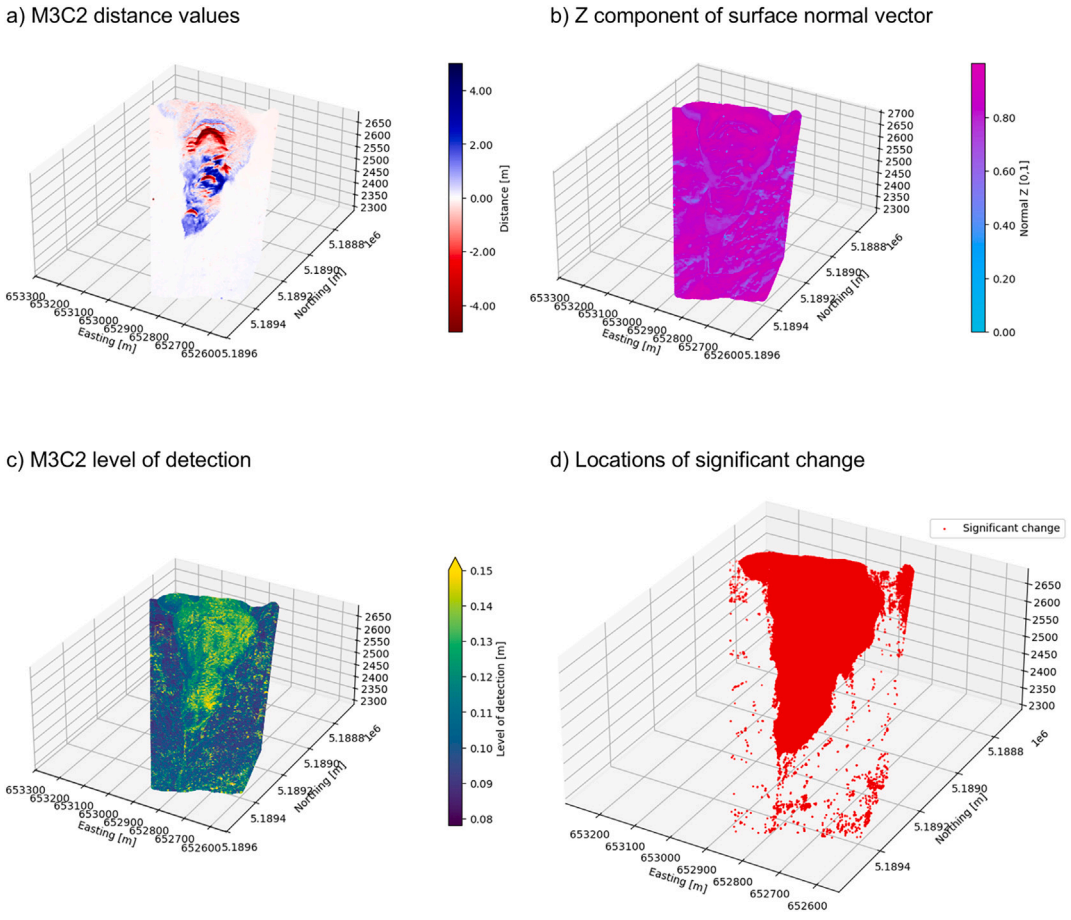

Fig. 3. Visualizations of change analysis results produced by the py4dgeo M3C2 algorithm, generated with the Python Matplotlib package with a) M3C2 distances showing magnitudes of surface change in the bitemporal rock glacier scene together with b) the Z component of the surface normal vector indicating whether the direction of change is rather vertical (value towards 1) or horizontal (value towards 0). Uncertainties associated with the quantified change are visualized in terms of level of detection in c), with d) significant change at those locations where the distance value exceeds the level of detection value.

3.2. Time series-based change analysis of highly dynamic scenes

In the second example, we feature a time series-based analysis demo using hourly 4D point clouds acquired by permanent Terrestrial Laser Scanning (TLS) at a sandy beach in Kijkduin, The Netherlands [29]. As a basis for these analyses, we load the LAS files of the time series into a py4dgeo SpatiotemporalAnalysis object. Using the force = True parameter creates a new analysis object on disk even if a file of the same name already exists, whereas the default force = False loads the existing object

```python
# generate a list of file paths to the point cloud files
import os
pc_dir = 'pointclouds' # 'pointclouds' is a placeholder that must be adapted to the actual data path
pc_list = os.listdir(pc_dir)

# create an empty SpatiotemporalAnalysis object (even if a file of this name already exists: force parameter)
analysis = py4dgeo.SpatiotemporalAnalysis('kijkduin.zip', force=True)

# specify the reference epoch from the list of point cloud files
reference_epoch_file = pc_list[0]

# read the reference epoch and set the timestamp from the list of timestamps
reference_epoch = py4dgeo.read_from_las(reference_epoch_file)
reference_epoch.timestamp = timestamps[0]

# set the reference epoch in the spatiotemporal analysis object
analysis.reference_epoch = reference_epoch

# specify corepoints, here all points of the reference epoch
analysis.corepoints = reference_epoch.cloud[::]

# customize direction algorithm inherited from the M3C2 algorithm class
class M3C2_Vertical(py4dgeo.M3C2):
    def directions(self):
        return np.array([0, 0, 1]) # vertical vector orientation

# specify M3C2 parameters for our custom algorithm class
analysis.m3c2 = M3C2_Vertical(cyl_radius=1.0, max_distance=10.0, registration_error = 0.019)

# collect epoch objects, start from pc_list[1:] to skip the reference epoch
epochs = []
for e, pc_file in enumerate(pc_list[1:]):
    epoch_file = os.path.join(pc_dir, pc_file)
    epoch = py4dgeo.read_from_las(epoch_file)
    epoch.timestamp = timestamps[e+1] # match enumerate index which starts from 1
    epochs.append(epoch)

# add epoch objects to the spatiotemporal analysis object
analysis.add_epochs(*epochs)

# average time series using rolling median filter in a temporal window defined as number of epochs
analysis.smoothed_distances = py4dgeo.temporal_averaging(analysis.distances,
    smoothing_window=14)
```

Listing 2. Code workflow to create a SpatiotemporalAnalysis object from a 3D time series.

without overwriting it (thus, base components such as reference epoch or M3C2 parametrization are protected from adaptation). Besides each point cloud dataset and corresponding timestamp information, this requires the specification of a reference epoch, here the first epoch of the time series. To calculate changes, a definition of core points and parametrization of the M3C2 algorithm are required. Since the beach topography is overall flat and dynamics are mostly deposition and removal of sand volumes, we customize the M3C2 algorithm to set all change directions purely vertically rather than computing the local surface orientation. Finally, we smooth the time series of vertical changes to remove noise (Listing 2).

The data structure can now be used to apply time series clustering following [20] as in Listing 3 (lines 1–8). This clusters the full change time series based on their similarity, by default using Euclidean distances, and provides insight into change patterns in the scene (Fig. 4a). For the beach application, yielded patterns are, e.g., sand accumulation through waves (sand bars), human activity involving sand removal at the dunes, or gradual accretion in different sections of the overall scene. Furthermore, 4D-OBCs can be extracted following [18] through parametrization of the time series change detection and region growing (Listing 3, lines 10–22). Resulting objects can be accessed for information about their time series properties, as well as their related change properties in the spatial

```python
# import kmeans clustering module from scikit-learn
from sklearn.cluster import KMeans

# use the smoothed distances for clustering
distances = analysis.smoothed_distances

# perform clustering for selected number of clusters (e.g., k=10)
kmeans = KMeans(n_clusters=10, random_state=0).fit(distances)

# parametrize the 4D-OBC extraction
algo = py4dgeo.RegionGrowingAlgorithm(
    window_width=14,
    minperiod=2,
    height_threshold=0.05,
    neighborhood_radius=1.0,
    min_segments=10,
    thresholds=[0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],
    seed_candidates=[15162]
)

# in case of re-runs, prior results need to be invalidated
analysis.invalidate_results(seeds=True, objects=True, smoothed_distances=False)

# run the algorithm
objects = algo.run(analysis)

# access info about a selected object from results
sel_idx = 0 # selects first object in the list
sel_object = objects[sel_idx]
# seed coordinate
seed_coord = analysis.corepoints.cloud[sel_object.seed.index]
# seed start and end epoch
start_epoch = sel_object.seed.start_epoch
end_epoch = sel_object.seed.end_epoch
# object plot of time series and spatial layout
sel_object.plot()
```

Listing 3. Code workflow to use a SpatiotemporalAnalysis object (Listing 2) for time series clustering (lines 1–8) and the extraction of 4D objects-by-change (lines 10–22).

domain. Fig. 4(b) shows an example object representing the temporary accumulation of a sand bar that was automatically detected in the time series and spatially delineated in the scene.

4. Impact

py4dgeo addresses an important gap in the processing and analysis of 3D/4D geographic point clouds by providing a scientific open-source Python library specifically tailored for change analysis workflows. The modular architecture and hybrid Python/C++ implementation enable automation of complex processing steps and promote user accessibility, allowing both domain experts and software developers to prototype, customize, and integrate novel methods within established workflows. This facilitates reproducible scientific studies and supports methodological development in 4D point cloud analysis. Its workflow compatibility through standard geospatial and Python data formats further enhances its utility by supporting multi-tool data processing pipelines.

The software implements recent methodological innovations in 4D geographic point cloud analysis [e.g.,7,15–19,30,31] and is being integrated in third-party funded projects (e.g., see funding sources) enabling dedicated software development in the form of new methods, extended functionalities, and support from professional research software engineers.

The open-source and community-driven nature of py4dgeo with an active development board fosters collaborative development, ensuring that the software evolves in response to emerging research challenges and user needs. By lowering the technical barriers to advanced 4D point cloud analysis, py4dgeo contributes to ongoing advances in 3D spatiotemporal data science and its applications across multiple domains.

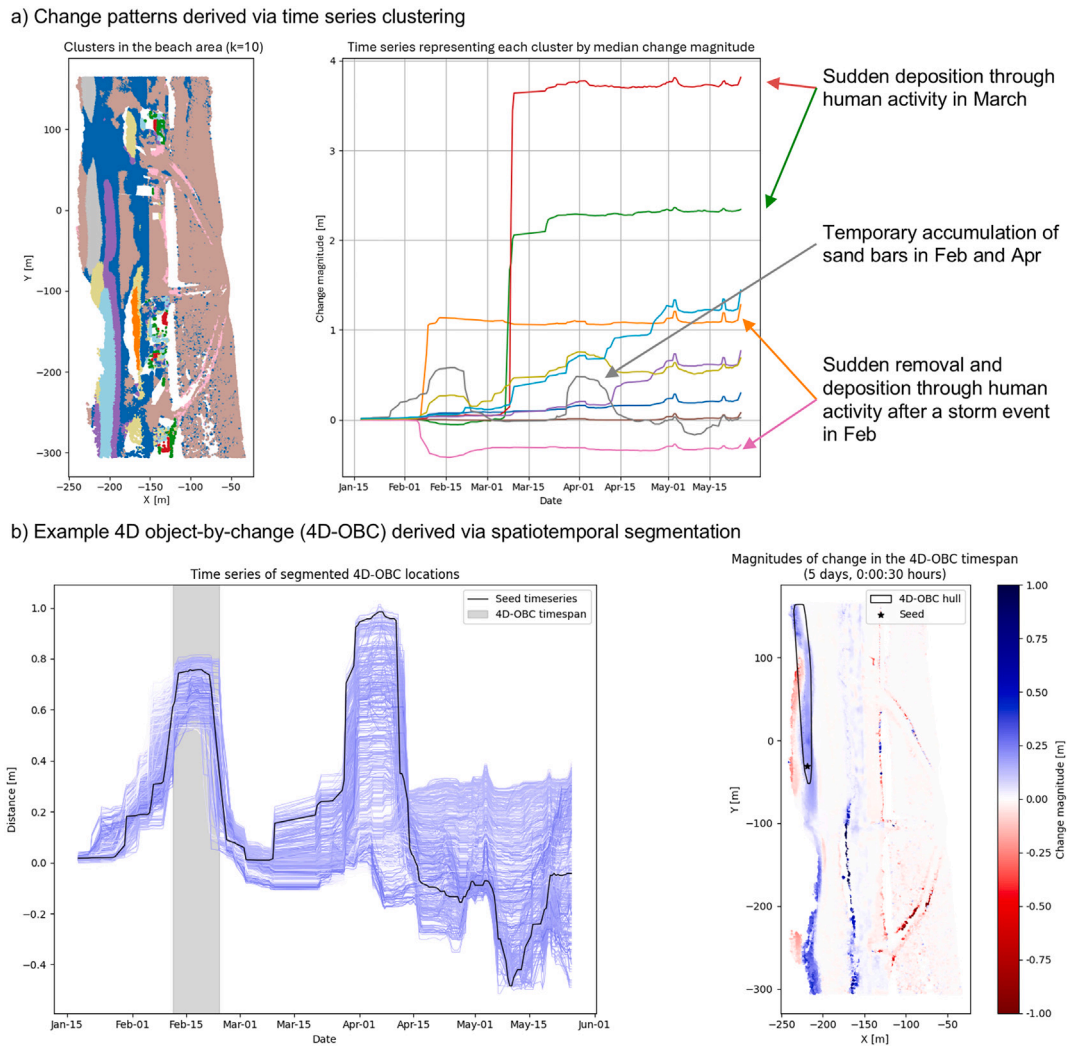

Fig. 4. Visualization of time series-based change analysis results. Results of time series clustering in (a) show change patterns in the scene colored by unique clusters with corresponding time series of surface changes (average per cluster). Visual assessment enables interpretation of characteristic process types (see example annotations on the right). Results of spatiotemporal segmentation in (b) show an example 4D object-by-change representing a sandbar that temporarily existed for several weeks (indicated by gray area in the time series). Only the time series similarity within that time span is considered for spatial delineation, which is depicted in the change scene on the right.).

5. Conclusions

py4dgeo offers an open-source Python framework for the analysis of 3D/4D geographic point clouds, integrating robust methods for registration, change detection, and time series-based analysis. Its design emphasizes workflow automation and adaptability, enabling reproducible and customizable scientific analyses. By supporting integration into broader geospatial workflows through standard data formats and community-driven development, py4dgeo provides a flexible foundation for multitemporal 3D and 4D change analysis.

Current workflows primarily operate on in-memory data structures, which may require spatial tiling or batching strategies when working with very large datasets or long time series. Future development will therefore focus on further improving scalability for large datasets, strengthening support for extensive multitemporal analyses, and expanding community-driven extensions. Through this, py4dgeo will remain a sustainable and adaptable platform for scientific analysis of 4D geographic point clouds.

CRediT authorship contribution statement

K. Anders: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Funding acquisition, Conceptualization. **D. Kempf:** Writing – review & editing, Validation, Supervision, Software, Methodology, Conceptualization. **W. Albert:** Writing – review & editing, Validation, Software, Methodology. **P. Andriushchenko:** Writing – review & editing, Validation, Software. **X. Huang:** Writing – review & editing, Software. **D. Hulskemper:** Writing – review & editing, Validation, Software, Methodology. **T. Isensee:** Writing – review & editing, Validation, Software. **D. Kapitan:** Writing – review & editing, Validation, Software. **R. Tabernig:** Writing – review & editing, Validation, Software, Methodology. **H. Weiser:** Writing – review & editing, Validation, Software, Methodology. **L. Winiwarter:** Writing – review & editing, Validation, Methodology, Conceptualization. **V. Zahs:** Writing – review & editing, Validation, Methodology. **B. Höfle:** Writing – review & editing, Validation, Supervision, Software, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The initial software development was supported by the Scientific Software Center (SSC) of Heidelberg University in the Open Call 2021. The Scientific Software Center is funded as part of the Excellence Strategy of the German Federal and State Governments. This work was supported by the Federal Ministry of Research, Technology and Space (BMFTR) – 02WDG1696 (AImon5.0 project). The BMFTR is funding the AImon5.0 project within the funding measure "Digital GreenTech – Environmental Engineering meets Digitalisation" as part of the "Research for Sustainability (FONA) Strategy". This work was further partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 535733258 (Extract4D project). K. Anders is grateful to the Baden-Württemberg Stiftung for the financial support through the Postdoctoral fellowship for leading early career researchers (CharAct4D project).

References

[1] Blanch X, Eltner A, Guinau M, Abellan A. Multi-epoch and multi-imagery (MEMI) photogrammetric workflow for enhanced change detection using time-lapse cameras. Remote Sens 2021;13(8):1460. https://doi.org/10.3390/rs13081460

[2] Lindenbergh R, Anders K, Campos M, Czerwonka-Schröder D, Höfle B, Kuschnerus M, Puttonen E, Prinz R, Rutzinger M, Voordendag A, Vos S. Permanent terrestrial laser scanning for near-continuous environmental observations: systems, methods, challenges and applications. ISPRS Open J Photogramm Remote Sens 2025;17:100094. https://doi.org/10.1016/j.ophoto.2025.100094

[3] Qin R, Tian J, Reinartz P. 3D change detection – approaches and applications. ISPRS J Photogramm Remote Sens 2016;122:41–56. https://doi.org/10.1016/j.isprsjprs.2016.09.013

[4] Czerwonka-Schröder D, Schulte F, Albert W, Hosseini K, Tabernig R, Yang Y, Höfle B, Holst C, Zimmermann K. Aimon5. 0-real-time monitoring of gravitational mass movements for critical infrastructure risk management with ai-assisted 3D metrology. In: Proceedings of the 6th joint international symposium on deformation monitoring (JISDM), karlsruhe, Germany; 2025. https://doi.org/10.5445/IR/1000179762

[5] Kromer RA, Abellán A, Hutchinson DJ, Lato M, Chanut MA, Dubois L, Jaboyedoff M. Automated terrestrial laser scanning with near-real-time change detection – monitoring of the Séchilienne landslide. Earth Surf Dynam 2017;5(2):293–310. https://doi.org/10.5194/esurf-5-293-2017

[6] Sharifisoraki Z, Dey A, Selzler R, Amini M, Green JR, Rajan S, Kwamena FA. Monitoring critical infrastructure using 3D lidar point clouds. IEEE Access 2023;11:314–36. https://doi.org/10.1109/ACCESS.2022.3232338

[7] Tabernig R, Albert W, Weiser H, Fritzmann P, Anders K, Rutzinger M, Höfle B. Temporal aggregation of point clouds improves permanent laser scanning of landslides in forested areas. Sci Remote Sens 2025;12:100254. https://doi.org/10.1016/j.srs.2025.100254

[8] Voordendag AB, Goger B, Klug C, Prinz R, Rutzinger M, Kaser G. Automated and permanent long-range terrestrial laser scanning in a high mountain environment: setup and first results. ISPRS Ann Photogramm Remote Sens Spatial Inf Sci 2021;V-2-2021:153–60, iSPRS-Annals. https://doi.org/10.5194/isprs-annals-V-2-2021-153-2021

[9] Eitel JUH, Höfle B, Vierling LA, Abellán A, Asner GP, Deems JS, Glennie CL, Joerg PC, LeWinter AL, Magney TS, Mandlburger G, Morton DC, Müller J, Vierling KT. Beyond 3-d: the new spectrum of lidar applications for earth and ecological sciences. Remote Sens Environ 2016;186:372–92. https://doi.org/10.1016/j.rse.2016.08.018

[10] Girardeau-Montaut D, Roux M, Marc R, Thibault G. Change detection on points cloud data acquired with a ground laser scanner. Int arch photogramm remote sens spat inf sci 2005;36(3):W19.

[11] Lague D, Brodu N, Leroux J. Accurate 3D comparison of complex topography with terrestrial laser scanner: application to the rangitikei canyon (n-z). ISPRS J Photogramm Remote Sens 2013;82:10–26. https://doi.org/10.1016/j.isprsjprs.2013.04.009

[12] CloudComPy Contributors. Cloudcompy: Python wrapper for cloudcompare; 2025. https://github.com/CloudCompare/CloudComPy [Accessed: 1 March 2026].

[13] Goelles T, Schlager B, Muckenhuber S, Haas S, Hammer T. 'Pointcloudset': efficient analysis of large datasets of point clouds recorded over time. J Open Source Softw 2021;6(65):3471. https://doi.org/10.21105/joss.03471

[14] Besl PJ, McKay ND. A method for registration of 3-D shapes. IEEE Trans Pattern Anal Mach Intell 1992;14(2):239–56.

[15] Yang Y, Schwieger V. Supervoxel-based targetless registration and identification of stable areas for deformed point clouds. J Appl Geod 2023;17(2):161–70. https://doi.org/10.1515/jag-2022-0031

[16] Winiwarter L, Anders K, Höfle B. M3c2-EP: pushing the limits of 3D topographic point cloud change detection by error propagation. ISPRS J Photogramm Remote Sens 2021. https://doi.org/10.1016/j.isprsjprs.2021.06.011

[17] Zahs V, Winiwarter L, Anders K, Williams JG, Rutzinger M, Höfle B. Correspondence-driven plane-based m3c2 for lower uncertainty in 3D topographic change quantification. ISPRS J Photogramm Remote Sens 2022;183:541–59. https://doi.org/10.1016/j.isprsjprs.2021.11.018

[18] Anders K, Winiwarter L, Mara H, Lindenbergh R, Vos SE, Höfle B. Fully automatic spatiotemporal segmentation of 3D lidar time series for the extraction of natural surface changes. ISPRS J Photogramm Remote Sens 2021. https://doi.org/10.1016/j.isprsjprs.2021.01.015

[19] Ulm M, Elias M, Eltner A, Lotsari E, Anders K. Automated change detection in photogrammetric 4D point clouds – transferability and extension of 4D objects-by-change for monitoring riverbank dynamics using low-cost cameras. Appl Geomat 2025;17(2):367–78. https://doi.org/10.1007/s12518-025-00623-9

[20] Kuschnerus M, Lindenbergh R, Vos S. Coastal change patterns from time series clustering of permanent laser scan data. Earth Surf Dynam 2021;9(1):89–103, eSurf. https://doi.org/10.5194/esurf-9-89-2021

[21] Frantzen P. Identifying high alpine geomorphological processes using permanent laser scanning time series [Thesis]. 2022. https://resolver.tudelft.nl/uuid:ce98c4e3-6ca1-4966-a5cf-2120f2fa44bf.

[22] Hulskemper D, Anders K, Antolínez JA, Kuschnerus M, Höfle B, Lindenbergh R. Characterization of morphological surface activities derived from near-continuous terrestrial lidar time series. Int Arch Photogramm Remote Sens Spat Inf Sci 2022. https://doi.org/10.5194/isprs-archives-XLVIII-2-W2-2022-53-2022

[23] Fritz J, Mesiti M, Thiele JP. A concise guide to good practices for automated testing and documentation of research software. Electron Commun EASST 2025;83. https://doi.org/10.14279/eceasst.v83.2622

[24] Guennebaud G, Jacob B, Avery P, Bachrach A, Barthelemy S. Eigen v3; 2010. http://eigen.tuxfamily.org.

[25] Blanco JL, Rai PK. Nanoflann: a c++ header-only fork of flann, a library for nearest neighbor (NN) with kd-trees; 2014. https://github.com/jlblancoc/nanoflann.

[26] Potůčková M, Albrechtová J, Anders K, Červená L, Dvořák J, Gryguc K, Höfle B, Hunt L, Lhotáková Z, Marcinkowska-Ochtyra A, Mayr A, Neuwirthová E, Ochtyra A, Rutzinger M, Šedová A, Šrollerů A, Kupková L. E-TRAINEE: open e-learning course on time series analysis in remote sensing. Int Arch Photogramm Remote Sens Spat Inf Sci 2023;XLVIII-1/W2-2023:989–96. https://doi.org/10.5194/isprs-archives-XLVIII-1-W2-2023-989-2023

[27] E-TRAINEE Development Team. E-TRAINEE - datasets; 2023. https://doi.org/10.5281/zenodo.10003575

[28] Hartl L, Zieher T, Bremer M, Stocker-Waldhuber M, Zahs V, Höfle B, Klug C, Cicoira A. Multi-sensor monitoring and data integration reveal cyclical destabilization of the Äußeres hochebenkar rock glacier. Earth Surf Dynam 2023;11(1):117–47, eSurf. https://doi.org/10.5194/esurf-11-117-2023

[29] Vos S, Anders K, Kuschnerus M, Lindenbergh R, Höfle B, Aarninkhof S, de Vries S. A high-resolution 4D terrestrial laser scan dataset of the kijkduin beach-dune system, the Netherlands. Sci Data 2022;9(1):191. https://doi.org/10.1038/s41597-022-01291-9

[30] Anders K, Winiwarter L, Höfle B. Improving change analysis from near-continuous 3D time series by considering full temporal information. IEEE Geosci Remote Sens Lett 2022. https://doi.org/10.1109/LGRS.2022.3148920

[31] Tabernig R, Albert W, Weiser H, Höfle B. A hierarchical approach for near real-time 3D surface change analysis of permanent laser scanning point clouds. In: Proceedings of the 6th joint international symposium on deformation monitoring (JISDM), karlsruhe, Germany; 2025. https://doi.org/10.5445/IR/1000180377