The Journal of Open Source Software

sensetrack: a python toolkit for remote-sensing imagery offset-tracking and preprocessing

Vincenzo Critelli[1], **Melissa Tondo**[1], **Cecilia Fabbiani**[1], **Francesco Lelli**[1], **Marco Mulas**[1], **and Alessandro Corsini**[1]

1 University of Modena and Reggio Emilia - Chemical and Geological Sciences Department

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

sensetrack is an open-source Python library designed to perform offset-tracking on geo-referenced imagery, with a specific focus on the detection and monitoring of surface displacements induced by landslide processes.
The library offers tools to preprocess and convert data from several satellite missions, including Sentinel-1, COSMO-SkyMed, and PRISMA, into geo-coded GeoTIFFs suitable for displacement analysis.

It provides an integrated and reproducible pipeline for image pair management, offset estimation using different algorithms (including phase correlation and optical flow), and output visualization or export.
sensetrack supports batch processing, modular workflows, and customization through XML-based processing graphs.

Statement of need

Landslides and mass movement processes pose a significant threat to infrastructure, settlements, and natural landscapes (Eisbacher, 1984; Froude & Petley, 2018; Klose, 2015; Mansour, 2011; Winter et al., 2016). Monitoring ground deformation in active or potentially unstable slopes is critical for risk mitigation and early warning.
While InSAR techniques have proven effective, offset-tracking provides complementary capabilities for detecting large, nonlinear, or fast-moving deformations that challenge conventional phase-based methods (Liu et al., 2025).

There is currently a lack of user-friendly, modular, and extensible Python libraries to support offset-tracking from various satellite platforms. sensetrack addresses this need by integrating image preprocessing, standardized conversion to geo-referenced formats, and multiple offset-tracking algorithms into a coherent workflow.

Unlike many existing tools for SAR-based displacement tracking that rely on Google Earth Engine (GEE), sensetrack runs entirely in a local Python environment. This design choice ensures full reproducibility, data privacy, and ease of integration in institutional or offline workflows.

Functionality and features

Offset-tracking module

The sensetrack.ot subpackage provides core functionalities for optical flow analysis, image normalization, interface management, and CLI for offset tracking. It is designed to work with satellite images and raster data, offering advanced algorithms and support tools for research and operational applications.

The `ot.interfaces.py` sub-module provides the foundational classes and utilities for managing images and implementing optical tracking algorithms within the project. At its core is the `Image` class, which encapsulates multi-band image data along with essential metadata such as georeferencing information, nodata handling, and band management. This class supports a variety of operations, including splitting images into individual bands, checking for coregistration between images, and accessing band-specific data, all while maintaining a consistent interface for both single-band and multi-band images. The design ensures that images are handled robustly, with automatic inference and management of nodata values and support for affine transformations and coordinate reference systems.

Complementing the image management functionality is the `OTAlgorithm` abstract base class, which serves as the blueprint for all offset tracking algorithms in the toolkit, implemented in `ot.algorithms.py` sub-module. It provides mechanisms for serializing and deserializing algorithm parameters from dictionaries, JSON, or YAML files, facilitating reproducibility and easy configuration. Additionally, it includes utility methods for converting pixel offsets into physical displacements, ensuring that results are meaningful in both pixel and real-world coordinates.

Implemented algorithms

1. `OpenCVOpticalFlow`
   The `algorithms.OpenCVOpticalFlow` algorithm provides a Python interface to the Farneback dense optical flow method (Horn & Schunck, 1981), as implemented in OpenCV's `calcOpticalFlowFarneback` function (Farnebäck, 2003). This approach estimates the motion field between two images by analyzing the apparent movement of pixel intensities, producing a dense displacement vector for every pixel. The core of the algorithm relies on constructing image pyramids, which allow it to capture both large and small displacements by progressively analyzing the images at multiple scales. At each level, the algorithm models local neighborhoods with polynomial expansions, enabling it to robustly estimate motion even in the presence of noise or textureless regions. The flexibility of the implementation allows users to fine-tune parameters such as the pyramid scale, window size, number of iterations, and the degree of smoothing, thus balancing accuracy and computational efficiency. After computing the flow, the results are transformed into images representing the horizontal and vertical components of the displacement, as well as the overall magnitude
2. `SkiOpticalFlowILK`
   The `algorithms.SkiOpticalFlowILK` (Lucas & Kanade, 1997) algorithm offers a Python interface to the Inverse Lucas-Kanade (ILK) method for dense optical flow estimation, as implemented in scikit-image's `optical_flow_ilk` function. This approach is designed to estimate the pixel-wise motion between two images by analyzing local intensity variations and tracking how small neighborhoods shift from the reference to the target image. The ILK method operates by minimizing the difference between the reference and the warped target image, iteratively refining the displacement field to achieve the best alignment. It is particularly well-suited for scenarios where the motion is relatively small and smooth, as it assumes that the displacement within each local window can be approximated linearly. The algorithm allows for customization of parameters such as the radius of the local window, the number of warping iterations, and the use of Gaussian smoothing or prefiltering, enabling users to adapt the method to different noise levels and image characteristics. After computing the displacement vectors, the results are transformed according to the affine properties of the target image, producing output images that represent the horizontal and vertical components of the motion, as well as the overall displacement magnitude
3. `SkiOpticalFlowTVL1`

The `algorithms.SkiOpticalFlowTVL1` (Zach et al., 2007) algorithm provides a Python interface to the TV-L1 optical flow method, as implemented in scikit-image's `optical_flow_tvl1` function. This approach is based on a variational framework that seeks to estimate the dense motion field between two images by minimizing an energy functional composed of a data attachment term and a regularization term. The TV-L1 method is particularly robust to noise and outliers, thanks to its use of the L1 norm for the data term and total variation (TV) regularization, which encourages piecewise-smooth motion fields while preserving sharp motion boundaries. The algorithm iteratively refines the displacement field through a multi-scale, coarse-to-fine strategy, allowing it to capture both large and small motions. Users can adjust parameters such as the strength of the data and regularization terms, the number of warping and optimization iterations, and the use of prefiltering, making the method adaptable to a wide range of imaging conditions. After the optical flow is computed, the results are mapped to the affine space of the target image, producing output images for the horizontal and vertical components of the displacement, as well as the overall magnitude

4. `SkiPCC_Vector`
   The `algorithms.SkiPCC_Vector` algorithm implements a phase cross-correlation (PCC) approach (Foroosh et al., 2002) for estimating local displacements between two images, leveraging the `phase_cross_correlation` function from scikit-image. Unlike traditional optical flow methods that rely on intensity gradients, this technique operates in the frequency domain. Since the base function `phase_cross_correlation` outputs a single displacement for two input arrays, this implementation provides an utility for splitting the two images into several sub-arrays in a rolling-window fashion (see the `stepped_rolling_window` help for further details), than `phase_cross_correlation` is performed for each pair of windows, and the results are collected in a dataframe-like structure where each record is associated with displacements in the two directions (fields `RSHIFT` and `CSHIFT` for row and column displacement respectively), the resultant displacement (`L2`), and the normalized root mean square deviation between analyzed moving windows (`NRMS`). By using phase normalization, the method enhances its sensitivity to translational differences while suppressing the influence of amplitude variations. The process can be further refined by adjusting the window size, step size, and upsampling factor, allowing for subpixel accuracy in the displacement estimates.

Command-line interface (CLI)

Each of the aformentioned algorithms can be executed through the command line. The CLI interface in this project serves as a flexible bridge between users and the core image processing algorithms, enabling command-line execution and configuration of complex workflows. At its foundation, the CLI is built around a generic base class that handles argument parsing, input validation, and algorithm instantiation. Each algorithm-specific module, such as those for OpenCV optical flow, phase cross-correlation, or scikit-image methods, extends this base class to introduce tailored command-line options reflecting the parameters and features of the underlying algorithm. Users interact with these modules by specifying arguments directly in the terminal, which are then parsed and mapped to the corresponding algorithm's configuration. The general workflow involves: 1. Parse command-line arguments. 2. Load reference and target images. 3. Coregistration 4. Preprocessing 5. Run the selected offset-tracking algorithm. 6. Export the displacement results to the specified output file. This design streamlines batch processing and reproducible analysis, allowing users to switch between different algorithms or parameter sets with minimal effort. The CLI modules that depend on cli.py inherit its structure, ensuring consistent behavior and a unified user experience across the toolkit. ## Additional sub-packages The `snap_gpt` sub-package is designed to facilitate the interaction with the SNAP Graph Processing Tool, a widely used platform for satellite image analysis. By providing programmatic access to SNAP's capabilities, this module enables users to automate complex processing chains, manage graph-based workflows, and integrate SNAP's advanced algorithms into custom remote sensing pipelines. Its architecture supports the orchestration of preprocessing, calibration, and product generation tasks, making it a valuable asset for large-scale and reproducible satellite data analysis.

The `sentinel` sub-package is specialized for handling data from the Sentinel satellite missions, which are part of the Copernicus program. It offers a comprehensive set of tools for reading, preprocessing, and analyzing Sentinel imagery, with routines tailored to the unique formats and metadata structures of these datasets. The module streamlines common operations such as radiometric correction, geometric alignment, and feature extraction, ensuring that users can efficiently prepare Sentinel data for further scientific or operational use.

The `prisma` sub-package focuses on the PRISMA hyperspectral satellite, providing dedicated functions for extracting and manipulating its spectral information. It supports the retrieval of hyperspectral cubes, metadata parsing, and the transformation of raw data into analysis-ready products.

Acknowledgements

This software was developed within the PARACELSO project, funded by the Italian Space Agency (ASI), with the goal of assisting local authorities in monitoring environmental risks and enhancing civil protection strategies. We gratefully acknowledge the support of the Italian authorities and institutions involved in territorial management and environmental monitoring.

References

Eisbacher, J. J., G. H. & Clague. (1984). *Destructive mass movements in high mountains: Hazard and management*. Geological Survey of Canada. https://doi.org/10.4095/120001

Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In J. Bigun & T. Gustavsson (Eds.), *Image analysis* (pp. 363–370). Springer Berlin Heidelberg. ISBN: 978-3-540-45103-7

Foroosh, H., Zerubia, J. B., & Berthod, M. (2002). Extension of phase correlation to subpixel registration. *IEEE Transactions on Image Processing*, *11*(3), 188–200. https://doi.org/10.1109/83.988953

Froude, M. J., & Petley, D. N. (2018). Global fatal landslide occurrence from 2004 to 2016. *Natural Hazards and Earth System Sciences*, *18*(8), 2161–2181. https://doi.org/10.5194/nhess-18-2161-2018

Horn, B. K. P., & Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*, *17*(1), 185–203. https://doi.org/https://doi.org/10.1016/0004-3702(81)90024-2

Klose, M. (2015). *Landslide databases as tools for integrated assessment of landslide risk*. Springer Cham. https://doi.org/10.1007/978-3-319-20403-1

Liu, Z., Peng, J., Su, H., Li, X., Wang, C., & Peng, Y. (2025). InSAR monitoring of large gradient deformation in coalfield and phase unwrapping research. *Geodesy and Geodynamics*. https://doi.org/https://doi.org/10.1016/j.geog.2025.06.001

Lucas, B., & Kanade, T. (1997). An iterative image registration technique with an application toStereo vision. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Vancouver*, *2*, 674–679. https://doi.org/10.5555/1623264.1623280

Mansour, N. R. &. M., M. F.; Morgenstern. (2011). Expected damage from displacement of slow-moving slides. *Landslides*, *8*, 117–131. https://doi.org/10.1007/s10346-010-0227-7

Winter, M. G., Shearer, B., Palmer, D., Peeling, D., Harmer, C., & Sharpe, J. (2016). The economic impact of landslides and floods on the road network. *Procedia Engineering*, *143*, 1425–1434. https://doi.org/https://doi.org/10.1016/j.proeng.2016.06.168

Zach, C., Pock, T., & Bischof, H. (2007). A duality based approach for realtime TV-L1 optical flow. *Pattern Recognition*, *4713*, 214–223. https://doi.org/10.1007/978-3-540-74936-3_22