

¹ `sensetrack`: a python toolkit for remote-sensing imagery offset-tracking and preprocessing

³ **Vincenzo Critelli**  ¹, **Melissa Tondo**  ², **Cecilia Fabbiani**  ², **Francesco Lelli**  ², **Marco Mulas**  ², and **Alessandro Corsini**  ²

⁵ **1** Po River Basin District Authority - Hydraulic and geological risk assessment and management area 2

⁶ **6** University of Modena and Reggio Emilia - Chemical and Geological Sciences Department

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- ⁷ [Review](#) 
- ⁸ [Repository](#) 
- ⁹ [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- ¹⁰ [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright¹⁰ and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#))¹¹.

Summary

¹² `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.

¹³ 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.

¹⁴ 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.

¹⁵ `sensetrack` supports batch processing, modular workflows, and customization through XML-based processing graphs.

Statement of need

¹⁶ Landslides and mass movement processes pose a significant threat to infrastructure, settlements, and natural landscapes ([Eibacher, 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.

¹⁷ 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](#)).

¹⁸ 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.

¹⁹ 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.

State of the field

²⁰ SenseTrack and DICpy are Python libraries for displacement estimation from image data that address different application domains. SenseTrack targets remote sensing and offset-tracking of remote sensing imagery (SAR and optical), integrating SNAP/GPT preprocessing and

39 georeferenced workflows, and relying on dense optical-flow and phase-correlation algorithms
40 implemented in OpenCV to process large scenes and produce GIS-ready displacement products.
41 In contrast, DICPy is designed for Digital Image Correlation in laboratory and engineering
42 experiments, focusing on subset- or global-based correlation methods to achieve high sub-pixel
43 accuracy in displacement and strain measurements on high-resolution speckle images.

44 Software design

45 The processing pipeline was conceived as a combination of independent tasks. In this
46 framework, SenseTrack's design emphasizes a modular architecture: subpackages (ot, snap_gpt,
47 sentinel, cosmo, prisma) expose interfaces for inputs, outputs, and algorithms, so individual
48 components can be swapped or extended without changing the overall processing chain.
49 Preprocessing routines, algorithm wrappers, and image-processing dispatchers are decoupled
50 from orchestration logic to maximize reuse, testability, and maintainability. The project
51 deliberately avoids dependencies with proprietary platforms and instead relies on open,
52 standardized tools such as SNAP-GPT, rasterio, h5py, and geopandas, enabling local execution,
53 full data control, reproducible pipelines, and easier deployment in diverse research and
54 operational environments.

55 Research impact statement

56 Offset-tracking methods based on image intensity, such as optical flow and windowed cross-
57 correlation, provide a complementary approach to phase-based InSAR techniques. While
58 InSAR enables highly precise deformation measurements under conditions of phase coherence,
59 offset tracking is less sensitive to decorrelation and can capture larger displacements, making
60 it suitable for rapidly moving or heterogeneous terrains. As a result, these methods are
61 particularly effective for large-area screening and monitoring of landslide activity, where they
62 can identify spatial patterns and kinematic trends at regional scale. When integrated with
63 InSAR analyses, offset-tracking approaches contribute to accelerating the development of
64 comprehensive knowledge frameworks that support hazard assessment and inform land-use
65 and territorial planning.

66 Functionality and features

67 Offset-tracking module

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

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

81 Complementing the image management functionality is the `OTAlgorithm` abstract base class,
82 which serves as the blueprint for all offset tracking algorithms in the toolkit, implemented
83 in `ot.algorithms.py` sub-module. It provides mechanisms for serializing and deserializing

84 algorithm parameters from dictionaries, JSON, or YAML files, facilitating reproducibility and
85 easy configuration. Additionally, it includes utility methods for converting pixel offsets into
86 physical displacements, ensuring that results are meaningful in both pixel and real-world
87 coordinates.

88 **Implemented algorithms**

89 1. `OpenCVOpticalFlow`

90 The `algorithms.OpenCVOpticalFlow` algorithm provides a Python interface to the
91 Farneback dense optical flow method (Horn & Schunck, 1981), as implemented in
92 OpenCV's `calcOpticalFlowFarneback` function (Farnebäck, 2003). This approach
93 estimates the motion field between two images by analyzing the apparent movement
94 of pixel intensities, producing a dense displacement vector for every pixel. The core of
95 the algorithm relies on constructing image pyramids, which allow it to capture both
96 large and small displacements by progressively analyzing the images at multiple scales.
97 At each level, the algorithm models local neighborhoods with polynomial expansions,
98 enabling it to robustly estimate motion even in the presence of noise or textureless
99 regions. The flexibility of the implementation allows users to fine-tune parameters such
100 as the pyramid scale, window size, number of iterations, and the degree of smoothing,
101 thus balancing accuracy and computational efficiency. After computing the flow, the
102 results are transformed into images representing the horizontal and vertical components
103 of the displacement, as well as the overall magnitude

104 2. `SkiOpticalFlowILK`

105 The `algorithms.SkiOpticalFlowILK` (Lucas & Kanade, 1997) algorithm offers a Python
106 interface to the Inverse Lucas-Kanade (ILK) method for dense optical flow estimation, as
107 implemented in scikit-image's `optical_flow_ilk` function. This approach is designed to
108 estimate the pixel-wise motion between two images by analyzing local intensity variations
109 and tracking how small neighborhoods shift from the reference to the target image. The
110 ILK method operates by minimizing the difference between the reference and the warped
111 target image, iteratively refining the displacement field to achieve the best alignment. It
112 is particularly well-suited for scenarios where the motion is relatively small and smooth,
113 as it assumes that the displacement within each local window can be approximated
114 linearly. The algorithm allows for customization of parameters such as the radius of the
115 local window, the number of warping iterations, and the use of Gaussian smoothing
116 or prefiltering, enabling users to adapt the method to different noise levels and image
117 characteristics. After computing the displacement vectors, the results are transformed
118 according to the affine properties of the target image, producing output images that
119 represent the horizontal and vertical components of the motion, as well as the overall
120 displacement magnitude

121 3. `SkiOpticalFlowTVL1`

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

137
138 4. `SkiPCC_Vector`
139 The `algorithms.SkiPCC_Vector` algorithm implements a phase cross-correlation (PCC)
140 approach ([Foroosh et al., 2002](#)) for estimating local displacements between two images,
141 leveraging the `phase_cross_correlation` function from `scikit-image`. Unlike traditional
142 optical flow methods that rely on intensity gradients, this technique operates in the
143 frequency domain. Since the base function `phase_cross_correlation` outputs a
144 single displacement for two input arrays, this implementation provides an utility for
145 splitting the two images into several sub-arrays in a rolling-window fashion (see the
146 `stepped_rolling_window` help for further details), than `phase_cross_correlation` is
147 performed for each pair of windows, and the results are collected in a dataframe-like
148 structure where each record is associated with displacements in the two directions
149 (fields `RSHIFT` and `CSHIFT` for row and column displacement respectively), the resultant
150 displacement (L_2), and the normalized root mean square deviation between analyzed
151 moving windows (NRMS). By using phase normalization, the method enhances its
152 sensitivity to translational differences while suppressing the influence of amplitude
153 variations. The process can be further refined by adjusting the window size, step size,
154 and upsampling factor, allowing for subpixel accuracy in the displacement estimates.

155 **Command-line interface (CLI)**

156 Each of the aforementioned algorithms can be executed through the command line. The CLI
157 interface in this project serves as a flexible bridge between users and the core image processing
158 algorithms, enabling command-line execution and configuration of complex workflows. At
159 its foundation, the CLI is built around a generic base class that handles argument parsing,
160 input validation, and algorithm instantiation. Each algorithm-specific module, such as those
161 for OpenCV optical flow, phase cross-correlation, or `scikit-image` methods, extends this base
162 class to introduce tailored command-line options reflecting the parameters and features of the
163 underlying algorithm. Users interact with these modules by specifying arguments directly in the
164 terminal, which are then parsed and mapped to the corresponding algorithm's configuration.
165 The general workflow involves:

- 166 1. Parse command-line arguments
- 167 2. Load reference and target images
- 168 3. Coregistration
- 169 4. Preprocessing
- 170 5. Run the selected offset-tracking algorithm
- 171 6. Export the displacement results to the specified output file

172 This design streamlines batch processing and reproducible analysis, allowing users to switch
173 between different algorithms or parameter sets with minimal effort. The CLI modules that
174 depend on `cli.py` inherit its structure, ensuring consistent behavior and a unified user experience
175 across the toolkit.

176 **Additional modules**

177 The `snap_gpt` module is designed to facilitate the interaction with the SNAP Graph Processing
178 Tool, a widely used platform for satellite image analysis. By providing programmatic access to
179 SNAP's capabilities, this module enables users to automate complex processing chains, manage
180 graph-based workflows, and integrate SNAP's advanced algorithms into custom remote sensing
181 pipelines. Its architecture supports the orchestration of preprocessing, calibration, and product
182 generation tasks, making it a valuable asset for large-scale and reproducible satellite data
183 analysis.

184 The `sentinel` module is specialized for handling data from the Sentinel satellite missions,
185 which are part of the Copernicus program. It offers a comprehensive set of tools for reading,
186 preprocessing, and analyzing Sentinel imagery, with routines tailored to the unique formats
187 and metadata structures of these datasets. The module streamlines common operations such

188 as radiometric correction, geometric alignment, and feature extraction, ensuring that users can
189 efficiently prepare Sentinel data for further scientific or operational use.

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

194 Acknowledgements

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

199 AI usage disclosure

200 Generative AI models were used to assist with bug fixing and editing selected portions of the
201 manuscript. The design, implementation, and testing of the software modules were performed
202 entirely by the authors.

203 References

- 204 Eibbacher, 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>
- 205 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
- 206 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>
- 207 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>
- 208 Horn, B. K. P., & Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*, 17(1), 185–203. [https://doi.org/10.1016/0004-3702\(81\)90024-2](https://doi.org/10.1016/0004-3702(81)90024-2)
- 209 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>
- 210 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/10.1016/j.geog.2025.06.001>
- 211 Lucas, B., & Kanade, T. (1997). An iterative image registration technique with an application to Stereo vision. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Vancouver*, 2, 674–679. <https://doi.org/10.5555/1623264.1623280>
- 212 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>
- 213 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/10.1016/j.proeng.2016.06.168>

- ²³⁰ 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

DRAFT