

# <sup>1</sup> `sensetrack`: a python toolkit for remote-sensing imagery offset-tracking and preprocessing

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

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

<sup>6</sup> **6** University of Modena and Reggio Emilia - Chemical and Geological Sciences Department

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

## Software

- <sup>7</sup> [Review](#) 
- <sup>8</sup> [Repository](#) 
- <sup>9</sup> [Archive](#) 

**Editor:** [Open Journals](#) 

**Reviewers:**

- <sup>10</sup> [@openjournals](#)

**Submitted:** 01 January 1970

**Published:** unpublished

## License

Authors of papers retain copyright<sup>11</sup> and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#))<sup>12</sup>.

## Summary

<sup>13</sup> `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.

<sup>14</sup> 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.

<sup>15</sup> 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.

<sup>16</sup> `sensetrack` supports batch processing, modular workflows, and customization through XML-based processing graphs.

## Statement of need

<sup>17</sup> 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.

<sup>18</sup> 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](#)).

<sup>19</sup> 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.

<sup>20</sup> 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.

## 35 Functionality and features

### 36 Offset-tracking module

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

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

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

### 57 Implemented algorithms

#### 58 1. OpenCVOpticalFlow

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

#### 73 2. SkiOpticalFlowILK

74 The `algorithms.SkiOpticalFlowILK` (Lucas & Kanade, 1997) algorithm offers a Python  
75 interface to the Inverse Lucas-Kanade (ILK) method for dense optical flow estimation, as  
76 implemented in scikit-image's `optical_flow_ilk` function. This approach is designed to  
77 estimate the pixel-wise motion between two images by analyzing local intensity variations  
78 and tracking how small neighborhoods shift from the reference to the target image. The  
79 ILK method operates by minimizing the difference between the reference and the warped  
80 target image, iteratively refining the displacement field to achieve the best alignment. It  
81 is particularly well-suited for scenarios where the motion is relatively small and smooth,  
82 as it assumes that the displacement within each local window can be approximated  
83 linearly. The algorithm allows for customization of parameters such as the radius of the  
84 local window, the number of warping iterations, and the use of Gaussian smoothing

85 or prefiltering, enabling users to adapt the method to different noise levels and image  
86 characteristics. After computing the displacement vectors, the results are transformed  
87 according to the affine properties of the target image, producing output images that  
88 represent the horizontal and vertical components of the motion, as well as the overall  
89 displacement magnitude

90 3. `SkiOpticalFlowTVL1`

91 The `algorithms.SkiOpticalFlowTVL1` (Zach et al., 2007) algorithm provides a  
92 Python interface to the TV-L1 optical flow method, as implemented in scikit-image's  
93 `optical_flow_tvl1` function. This approach is based on a variational framework  
94 that seeks to estimate the dense motion field between two images by minimizing an  
95 energy functional composed of a data attachment term and a regularization term. The  
96 TV-L1 method is particularly robust to noise and outliers, thanks to its use of the  
97 L1 norm for the data term and total variation (TV) regularization, which encourages  
98 piecewise-smooth motion fields while preserving sharp motion boundaries. The algorithm  
99 iteratively refines the displacement field through a multi-scale, coarse-to-fine strategy,  
100 allowing it to capture both large and small motions. Users can adjust parameters  
101 such as the strength of the data and regularization terms, the number of warping and  
102 optimization iterations, and the use of prefiltering, making the method adaptable to  
103 a wide range of imaging conditions. After the optical flow is computed, the results  
104 are mapped to the affine space of the target image, producing output images for the  
105 horizontal and vertical components of the displacement, as well as the overall magnitude  
106

107 4. `SkiPCC_Vector`

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

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

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

- 135 1. Parse command-line arguments
- 136 2. Load reference and target images
- 137 3. Coregistration

138     4. Preprocessing  
139     5. Run the selected offset-tracking algorithm  
140     6. Export the displacement results to the specified output file  
141     This design streamlines batch processing and reproducible analysis, allowing users to switch  
142     between different algorithms or parameter sets with minimal effort. The CLI modules that  
143     depend on `cli.py` inherit its structure, ensuring consistent behavior and a unified user experience  
144     across the toolkit.

## 145     Additional modules

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

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

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

## 163     Acknowledgements

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

## 168     References

- 169     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>
- 170     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
- 171     Foroosh, H., Zerubia, J. B., & Berthod, M. (2002). Extension of phase correlation to  
172     subpixel registration. *IEEE Transactions on Image Processing*, 11(3), 188–200. <https://doi.org/10.1109/83.988953>
- 173     Froude, M. J., & Petley, D. N. (2018). Global fatal landslide occurrence from 2004 to 2016.  
174     *Natural Hazards and Earth System Sciences*, 18(8), 2161–2181. <https://doi.org/10.5194/nhess-18-2161-2018>
- 175     Horn, B. K. P., & Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*,  
176     17(1), 185–203. [https://doi.org/10.1016/0004-3702\(81\)90024-2](https://doi.org/10.1016/0004-3702(81)90024-2)

- 182 Klose, M. (2015). *Landslide databases as tools for integrated assessment of landslide risk*.  
183 Springer Cham. <https://doi.org/10.1007/978-3-319-20403-1>
- 184 Liu, Z., Peng, J., Su, H., Li, X., Wang, C., & Peng, Y. (2025). InSAR monitoring of large  
185 gradient deformation in coalfield and phase unwrapping research. *Geodesy and Geodynamics*.  
186 <https://doi.org/https://doi.org/10.1016/j.geog.2025.06.001>
- 187 Lucas, B., & Kanade, T. (1997). An iterative image registration technique with an application  
188 to Stereo vision. *Proceedings of the Seventh International Joint Conference on Artificial  
189 Intelligence, Vancouver*, 2, 674–679. <https://doi.org/10.5555/1623264.1623280>
- 190 Mansour, N. R. & M., M. F.; Morgenstern. (2011). Expected damage from displacement of  
191 slow-moving slides. *Landslides*, 8, 117–131. <https://doi.org/10.1007/s10346-010-0227-7>
- 192 Winter, M. G., Shearer, B., Palmer, D., Peeling, D., Harmer, C., & Sharpe, J. (2016). The  
193 economic impact of landslides and floods on the road network. *Procedia Engineering*, 143,  
194 1425–1434. <https://doi.org/https://doi.org/10.1016/j.proeng.2016.06.168>
- 195 Zach, C., Pock, T., & Bischof, H. (2007). A duality based approach for realtime TV-L1 optical  
196 flow. *Pattern Recognition*, 4713, 214–223. [https://doi.org/10.1007/978-3-540-74936-3\\_22](https://doi.org/10.1007/978-3-540-74936-3_22)

DRAFT