Documentation

PixelMatching

align_images(image1, image2, reference_area: gpd.GeoDataFrame, number_of_control_points: int, cell_size: int = 40, tracking_area_size: int = 60, select_bands=None, image_alignment_via_lsm=False):

Aligns two georeferenced images opened in rasterio by matching them in the area given by the reference area. In areas, where only one of the two images contains data, the matrix values will be set to 0 in both images.

Parameters

---

image1 :
A raster image opened in the rasterio package to be aligned with the second image.

image2 :
A raster image opened in the rasterio package to be aligned with the first image. The alignment takes place via an adjustment of the transform of the second raster image, which means that the first image is assumed to be correctly georeferenced.

reference_area : gpd.GeoDataFrame
A single-element GeoDataFrame, containing a polygon for specifying the reference area used for the alignment. This is the area, where no movement is suspected.

number_of_control_points: int
An approximation of how many points should be created on the reference_area polygon to track possible camera position differences. For details see grid_points_on_polygon.

cell_size: int = 40
The size of image sections in pixels which are compared during the tracking. See track_movement for details.

tracking_area_size: int = 60
The size of the image sections in pixels which are used as a search area during the tracking. Must be greater than the parameter cell_size. See track_movement for details.

select_bands = None
For multi-channel images the channel, which should be used for tracking. If it is None (the default) it will select the first three channels. If it is an integer, only the respective single channel will be selected. For multi-channel selection a list can be provided (e.g. [0,2,3] for the first, third and fourth channel of the given raster image). Channels are assumed to be in the same order for the two provided image files.

image_alignment_via_lsm: bool = False
If False, each control point is being tracked using the cross-correlation approach and their average displacement is taken into account (see also track_cell). If True, after calculating the average displacement, the least-squares approach is used on the whole reference area, to determine the exact transformation matrix necessary to align the two images optimally. For

large images, this step is cost-intensive and can take some time.

Returns

[image1_matrix, new_matrix2, image_transform]: The two matrices representing the raster image as numpy arrays and their transform for the coordinate reference system of the input images. As the two matrices are aligned, they possess the same transformation.

track_movement(image1_matrix, image2_matrix, image_transform, tracking_area: gpd.geodataframe, number_of_tracked_points: int, cell_size: int = 40, tracking_area_size: int = 50, tracking_method: str = "lsm", remove_outliers: bool = True, retry_matching: bool = True):

Calculates the movement of points between two aligned raster image matrices (with the same transform) in a given area using the cross-correlation or the least-squares approach.

Parameters

---

image1_matrix :
A numpy array with 2 or three dimensions, where the last two dimensions give the image height and width respectively and for a threedimensional array, the first dimension gives the channels of the raster image. This array should represent the earlier observation.

image2_matrix :
A numpy array of the same format as image1_matrix representing the second observation.

image_transform :
An object of the class Affine as provided by the rasterio package. The two images are assumed to be aligned (for example as a result of align_images) and therefore have the same transform.

tracking_area : gpd.GeoDataFrame
A GeoDataFrame containing a single polygon, where pixels should be tracked. In the area specified by this polygon, a regularly spaced grid of points is created using grid_points_on_polygon. The polygon is assumed to have the same coordinate reference system as the raster images (i.e. it is compatible with the given image_transform).

number_of_tracked_points : int
The number of points to be created by grid_points_on_polygon (for details see there), which is the number of separate pixels that are being tracked.

cell_size : int = 40
The size of the cells in pixels, which will be created in order to compare the two images. The function get_submatrix_symmetric is used for extracting the image section based on this value. This parameter determines the size ob detectable object as well as the influence of boundary effects.

tracking_area_size : int = 50
The size of the area in pixels, where fitting image sections are being searched. This parameter determines the maximum detectable movement rate and influences computation speed. This value must be higher than the parameter cell_size.

tracking_method : str = "lsm"
One of "lsm" (for least-squares matching) or "cross-correlation", determining the method used for tracking each cell.

remove_outliers : bool = True
Determines if outliers determined via large deviations in velocity and movement direction will be removed (for details see remove_outlying_tracked_pixels). Low correlation matchings (cross-correlation coefficient < 0.5) and matchings with a transformation determinant < 0.8 or > 1.2 will always be removed.

retry_matching : bool = True
When the first matching using the least-squares approach was unsuccessful (e.g. due to a low correlation coefficient for the initial values, a non-converging optimization problem or because the pixel was removed as an outlier), it can be rerun using a different starting value. If this parameter is True, the already calculated average movement rate for its adjacent pixels will be used as initial value for the least-squares optimization problem.

Returns

---

tracked_pixels: A DataFrame containing one row for every tracked pixel, specifying the position of the tracked pixel (in terms of matrix indices) and the movement in x- and y-direction in pixels. Invalid matchings are marked by NaN values for the movement.

retrack_wrong_matching_pixels(tracked_pixels, track_matrix, search_matrix, cell_size, tracking_area_size, fallback_on_cross_correlation = False):

Tries to retrack all pixels with movement given by np.nan via the least-squares approach.

Parameters

---

tracked_pixels :
A DataFrame containing the tracked pixels with their position and movement rates as returned by track_movement.

track_matrix :
The raster image matrix of the first image.

search_matrix :
The raster image matrix of the second image. The two matrices are assumed to be aligned (for example using align_images).

cell_size : int
The cell size used for tracking. For details see track_movement.

tracking_area_size : int
The size of the area in which the corresponding image section will be searched. For details see track_movement.

fallback_on_cross_correlation : bool = False
If True, the function will return the matching result from the cross-correlation approach, if the least-squares approach did not find a valid matching in the second try.

Returns

---

tracked_pixels: A DataFrame containing one row for every tracked pixel, specifying the position of the tracked pixel (in terms of matrix indices) and the movement in x– and y–direction in pixels. Invalid matchings are marked by NaN values for the movement.

remove_outlying_tracked_pixels(tracked_pixels, from_rotation=False, from_velocity=False):

Removes outliers given by deviations in velocity or movement direction from a given dataframe of tracked pixels.

Parameters

---

tracked_pixels :
A DataFrame containing the tracked pixels with their position and movement rates as returned by track_movement.

from_rotation : bool = True
If True, pixels with a movement direction which differs more than 90° from the average movement direction of its adjacent pixels, will be considered outliers and their movement rates will be set to NaN.

from_velocity : bool = True
If True, pixels with a movement rate more than twice the average movement rate of its adjacent pixels, will be considered outliers and their movement rates will be set to NaN.

Returns

---

tracked_pixels: A DataFrame containing one row for every tracked pixel, specifying the position of the tracked pixel (in terms of matrix indices) and the movement in x– and y–direction in pixels. Invalid matchings are marked by NaN values for the movement.

get_tracked_pixels_square(tracked_pixels, central_pixel_coordinates):

Returns a dataframe which consists of the eight (or less) surrounding pixels, for a given pixel with matrix indices central_pixel_coordinates. This is a helper function for retrack_wrong_matching_pixels and remove_outlying_tracked_pixels

Parameters

---

tracked_pixels :
A DataFrame containing the tracked pixels with their position and movement rates as returned by track_movement.

central_pixel_coordinates
The coordinates (as matrix index tuple) of the central pixel, for which the adjacent pixels will be returned.

Returns

---

neighbouring_tracked_pixels: A DataFrame containing all available adjacent pixels with their position and movement rates. The central pixel is not included.

track_cell_lsm(tracked_cell_matrix: np.ndarray, search_cell_matrix: np.ndarray, initial_shift_values = None, return_full_coefficients=False):

Calculates the transformation parameters for a given image cell to match optimally the corresponding section in the second image using the least-squares approach.

Parameters

---

tracked_cell_matrix : np.ndarray
The image section of the first image that will be matched to a part of the search image section.

search_cell_matrix : np.ndarray
The image section of the second, which is used as a search area for the image section from the first image. This array needs to be larger than the tracked_cell_matrix.

initial_shift_values = None
If None, the initial values for the shift between the images are being calculated from the cross-correlation approach using the function track_cell. Otherwise, the provided values will be used as initial shift values. The transformation matrix is always initialized as the identity matrix $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

return_full_coefficients : bool = False
If True, returns the six parameters for the affine transformation, instead of the movement rates for the central pixel of tracked_cell_matrix.

Returns

[**shift_rows, shift_columns**]: The y- and x-movement (in pixels) for the central pixel of tracked_cell_matrix. If return_full_coefficients == True, returns instead [**t1,t2,t3,t4,b1,b2**], where t1, t2, t3, t4 are the entries of the transformation matrix and b1, b2 are the entries of the shift vector. If the matching was not successful, return NaNs instead.

lsm_loss_function_rotation(coefficients, tracked_cell_matrix: np.ndarray, interpolator_search_cell, central_indices, indices):

Calculates the loss of the optimization problem for the least-squares approach for given coefficients.

Parameters

---

coefficients
A list of six integers [t1,t2,t3,t4,b1,b2] specifying the transformation parameters as in track_cell_lsm and the least-squares approach.

tracked_cell_matrix : np.ndarray
The given image section from the first image to be compared to an image section of the second image, which is given by interpolator_search_cell.

interpolator_search_cell :
The interpolator for the section from the second image. An object of the class RegularGridInterpolator from scipy.interpolate. For details see move_cell_rotation_approach.

central_indices
The index of the centre pixel of the tracked image cell, expressed in terms of matrix indices of the search cell matrix (from the second image). For details see move_cell_rotation_approach.

indices

A 2-dimensional array, with two rows and number of columns equal to the number of entries in tracked_cell_matrix, containing the indices of the pixels of tracked_cell_matrix expressed in terms of the search cell matrix. For details see move_cell_rotation_approach.

Returns

---

loss: A float, giving the loss for the provided transformation parameters

move_cell_rotation_approach(coefficients, tracked_cell_matrix_shape, interpolator_search_cell, central_indices, indices):

For given transformation coefficients and indices, returns the section of the search_cell with size given by tracked_cell_matrix_shape, obtained by applying the transformation to the indices.

Parameters

---

coefficients :

A list of six integers [t1,t2,t3,t4,b1,b2] specifying the transformation parameters as in track_cell_lsm.

tracked_cell_matrix_shape :

The shape of the tracked image section (from the first image). This is used to get a section of equal size from the second image, using the interpolator_search_cell.

interpolator_search_cell :

The interpolator for the search cell from the second image, used to obtain values for non-integer results of the application of the affine transform to the indices. As in lsm_loss_function_rotation, this has to be an object of the class RegularGridInterpolator.

indices :

A 2-dimensional array, with two rows and number of columns equal to the number of entries in tracked_cell_matrix, containing the indices of the pixels of tracked_cell_matrix expressed in terms of the search cell matrix. The affine transformation given by the coefficients is applied to these indices and the interpolated search cell is evaluated at the transformed indices.

Returns

---

moved_cell_matrix: A numpy array of the shape given by tracked_cell_matrix_shape, which represents the section of the second image, moved according to the affine transform coefficients with respect to the search cell.

track_cell

Calculates the movement of an image section using the cross-correlation approach.

Parameters

---

tracked_cell_matrix: np.ndarray

An array (a section of the first image), which is compared to sections of the search_cell_matrix (a section of the second image).

search_cell_matrix: np.ndarray

An array, which delimits the area in which possible matching image sections are searched.

Returns

---

movement_for_best_correlation: A two-element list, giving the movement rates in x- and y-direction respectively.

get_submatrix_symmetric(central_index, shape, matrix):

Extracts a symmetric section of a given matrix and shape, so that central_index is in the centre of the returned array. If shape specifies an even height or width, it is decreased by one to ensure that there exists a unique central index in the returned array

Parameters

---

central_index :
A two-element list, containing the row and column indices of the entry, which lies in the centre of the returned array.

shape :
A two-element list, containing the row and column number of the returned array. If one of these is an even number, it will be decreased by one to ensure that a unique central index exists.

matrix :
The matrix from which the section is extracted.

Returns

---

submatrix: A numpy array of the specified shape.

HandleGeometries

grid_points_on_polygon(polygon: gpd.GeoDataFrame, number_of_points: int = 10):

Creates an evenly spaced grid of points inside the given polygon. An approximation of the number of created points can be given, the actual number of points may differ depending on the shape of the polygon. The resulting GeoDataFrame will have the same coordinate reference system as the polygon.

Parameters

---

polygon: gpd.GeoDataFrame
The polygon where the points will be created.

number_of_points: int = 10
The approximate number of points to be created. The function calculates an approximate spacing based on this number and the area ratio of the given polygon and its enclosing rectangle so that the resulting grid is exactly evenly spaced and contains roughly this number of points ($\pm \sim 3\%$).

Returns

---

points: A GeoDataFrame containing the created points.

get_raster_indices_from_points

Transforms the coordinates of points in a given coordinate reference system to their respective matrix indices for a given transform

Parameters

---

points: gpd.GeoDataFrame
A GeoDataFrame containing points in a certain coordinate reference system.

raster_matrix_transform
An object of the class Affine as used by the rasterio package, representing the transform from the matrix indices to the coordinate reference system of the points.

Returns

---

rows, cols: The row and column indices respectively for the points.

get_overlapping_area(file1, file2):

Crops the two files to their intersection and pads multi-channel images with different pixel sizes with 0 so that the resulting matrices have the same size.

Parameters

---

file1, file2: The two raster image files as opened rasterio objects.

Returns

[**array_file1,**
array_file2]: The raster matrix for the first file and its respective transform.

[**array_file2, array_file2_transform**]: The raster matrix for the first file and its respective transform.

georeference_tracked_points

Georeferences a DataFrame with tracked points and calculates their movement (absolute and per year) in the unit specified by the coordinate reference system.

Parameters

---

tracked_pixels: pd.DataFrame
A DataFrame containing tracked pixels with columns "row", "column" (specifying the position of the point on the raster image), and "movement_row_direction", "movement_column_direction", "movement_distance_pixels" (specifying its movement in terms of raster pixels).

raster_transform:
An object of the class Affine as used by the rasterio package, representing the transform from the matrix indices to the coordinate reference system of the points.

crs:
An identifier for a coordinate reference system to which the resulting GeoDataFrame will be projected.

years_between_observations = 1

A float representing the number of years between the two images for calculating average yearly movement rates.

Returns

---

georeferenced_tracked_pixels:

A GeoDataFrame containing the tracked pixels with the previously mentioned columns, as well as the columns "movement_distance" and "movement_distance_per_year", specifying the movement in the unit of the given coordinate reference system and one geometry column.

HandleFiles

read_two_image_files(path1: str, path2: str):

Reads two raster images using the rasterio package from the given paths

path1, path2: str

The paths to the two raster image files.

Returns

---

file1, file2: The two files opened in the rasterio package

write_results(georeferenced_tracked_pixels: gpd.GeoDataFrame, parameter_dict, folder_path: str):

Writes the results of a performed tracking to a specified folder along with a file containing the parameters of this tracking. The files are being saved with the name "tracking_results", thus it is recommended to use a different folder for each individual tracking.

Parameters

---

georeferenced_tracked_pixels: gpd.GeoDataFrame

A GeoDataFrame containing the results of the tracking. This will be saved to "folder_path/tracking_results.geojson" and "folder_path/tracking_results.csv" respectively.

parameter_dict

A dictionary containing the parameters used for this run. Can contain additional notes regarding the tracking and will be saved to "folder_path/tracking_parameters.txt".

folder_path: str

The location of the folder, where the results should be saved. Not existing directories will be created.

Returns

None

read_tracking_results(file_path: str):

Reads results of a previous tracking to a GeoDataFrame.

Parameters

---

file_path: str
The path to read the file from. The file is expected to be readable for GeoPandas (e.g. a geojson file)

Returns

---

georeferenced_tracked_pixels: A GeoDataFrame containing the results of a previous tracking.

MakePlots

plot_raster_and_geometry(raster_matrix: np.ndarray, raster_transform, geometry: gpd.GeoDataFrame, alpha=0.6):

Plots a matrix representing a raster image with given transform and the geometries of a given GeoDataFrame in one figure.

Parameters

---

raster_matrix: np.ndarray
The matrix representing the raster image to be plotted.

raster_transform
An object of the class Affine as used by the rasterio package, which gives the transform of the raster image to the coordinate reference system of the geometry GeoDataFrame.

geometry: gpd.GeoDataFrame
The geometry to be plotted.

alpha=0.6
The opacity of the plotted geometry (which will be plotted on top of the raster image).

Returns

None

plot_movement_of_points(raster_matrix: np.ndarray, raster_transform, point_movement: gpd.GeoDataFrame, masking_polygon: gpd.GeoDataFrame = None):

Plots the movement of tracked points as a geometry on top of a given raster image matrix. Velocity is shown via a colour scale, while the movement direction is shown with arrows for selected pixels.

Parameters

---

raster_matrix: np.ndarray
The matrix representing the raster image to be plotted.

raster_transform :
An object of the class Affine as used by the rasterio package, which gives the transform of the raster image to the coordinate reference system of the geometry GeoDataFrame.

point_movement: gpd.GeoDataFrame

A GeoDataFrame containing the columns "row", "column" giving the position of the points expressed in matrix indices, as well as "movement_column_direction", "movement_row_direction" and "movement_distance_per_year". The unit of the movement is taken from the coordinate reference system of this GeoDataFrame.

masking_polygon: gpd.GeoDataFrame = None

A single-element GeoDataFrame to allow masking the plotted points to a certain area. If None, the points will not be masked.

show_figure : bool = True

If True, the created plot is displayed on the current canvas.

save_path : str = None

The file location, where the created plot is stored. When no path is given (the default), the figure is not saved.

Returns

None