4D change analysis of near-continuous LiDAR time series for applications in geomorphic monitoring

Time series-based change analysis of surface dynamics at a sandy beach

In this practical, you will perform time series-based surface change analysis on a time series of permanent TLS point clouds of the sandy beach at Kijkduin for a timespan of around 6 months (Vos et al., 2022).

The objective is to assess surface dynamics with two methods: time series clustering (following Kuschnerus et al., 2021) and 4D objects-by-change (following Anders et al., 2021). Look into the related articles for comparison of possible surface dynamics at the site and help for deciding on suitable parameters, etc.

Software and data

This exercise should be solved using Python with the `py4dgeo` library. The workflow will be introduced throughout this practical. Also, make use of the software documentations!

Use CloudCompare or GIS Software (e.g., QGIS) to check and visualize your results.

The dataset is a subsampled version of the original time series, using 12-hourly epochs of point clouds and spatial subsampling to 50 cm. In the data directory `kijkduin`, you find the prepared input point clouds and a core points point cloud, which is manually cleaned from noise.

Loading data and calculation of surface changes

Prepare the analysis by compiling the list of files (epochs) and read the timestamps from the file names (format `YYMMDD_hhmmss`) into `datetime` objects. Use the point cloud files and timestamps to create a py4dgeo `SpatiotemporalAnalysis` object. For this you need to instantiate the M3C2 algorithm. You can use the point cloud file `170115_150816_aoi_50cm.laz` as core points. Explore the point cloud properties in CloudCompare:

- Considering the available point density and surface characteristics, what would be a suitable cylinder radius for the distance calculation?

- What would be a suitable approach to derive the surface normals in this topography and expected types of surface changes?

Hint: In this flat topography and predominant provess of sand deposition and erosion, it can be suitable to orient the normals purely vertically. In this case, they do not need to be computed, and you can customize the py4dgeo algorithm accordingly.

Use the first point cloud in the time series (list of files) as reference epoch. You can assume a registration error of 1.9 cm for the M3C2 distance calculation (cf. Vos et al., 2022).

Explore the spatiotemporal change information by visualizing the changes at a selected epoch and visualizing the time series at a selected location.

First, we start by setting up the Python environment and data:

In [1]:
```
# import required modules
import os
import numpy as np
import py4dgeo
from datetime import datetime

# specify the data path
data_path = 'path-to-data'

# check if the specified path exists
if not os.path.isdir(data_path):
    print(f'ERROR: {data_path} does not exist')
    print('Please specify the correct path to the data directory by replacing <path

# sub-directory containing the point clouds
pc_dir = os.path.join(data_path, 'pointclouds')

# list of point clouds (time series)
pc_list = os.listdir(pc_dir)
pc_list[:5] # print the first elements
```

Out[1]:
```
['kijkduin_170117_120041.laz',
 'kijkduin_170118_000050.laz',
 'kijkduin_170120_120036.laz',
 'kijkduin_170121_000046.laz',
 'kijkduin_170121_120055.laz']
```

In the list of point cloud files you can see that we have one laz file per epoch available. The file name contains the timestamp of the epoch, respectively, in format `YYMMDD_hhmmss`. To use this information for our analysis, we read the timestamp information from the file names into `datetime` objects.

In [2]:
```
# read the timestamps from file names
timestamps = []
for f in pc_list:
    if not f.endswith('.laz'):
        continue
    # get the timestamp from the file name
    timestamp_str = '_'.join(f.split('.')[0].split('_')[1:]) # yields YYMMDD_hhmmss

    # convert string to datetime object
    timestamp = datetime.strptime(timestamp_str, '%y%m%d_%H%M%S')
    timestamps.append(timestamp)

timestamps[:5]
```

```
Out[2]: [datetime.datetime(2017, 1, 17, 12, 0, 41),
         datetime.datetime(2017, 1, 18, 0, 0, 50),
         datetime.datetime(2017, 1, 20, 12, 0, 36),
         datetime.datetime(2017, 1, 21, 0, 0, 46),
         datetime.datetime(2017, 1, 21, 12, 0, 55)]
```

Now we use the point cloud files and timestamp information to create a `SpatiotemporalAnalysis` object

```
In [3]: analysis = py4dgeo.SpatiotemporalAnalysis(f'{data_path}/kijkduin.zip', force=True)
```

```
[2023-03-29 15:54:53][INFO] Creating analysis file I:\etrainee_data\kijkduin/kijkdui
n.zip
```

As reference epoch, we use the first epoch in our time series:

```
In [4]: # specify the reference epoch
        reference_epoch_file = os.path.join(pc_dir, pc_list[0])

        # read the reference epoch and set the timestamp
        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
```

```
[2023-03-29 15:54:53][INFO] Reading point cloud from file 'I:\etrainee_data\kijkduin
\pointclouds\kijkduin_170117_120041.laz'
[2023-03-29 15:54:53][INFO] Building KDTree structure with leaf parameter 10
[2023-03-29 15:54:53][INFO] Saving epoch to file 'C:\Users\k53\AppData\Local\Temp\tm
p1p2qu55j\reference_epoch.zip'
```

For epochs to be added, we now configure the M3C2 algorithm to derive the change values. We would like to set the normals purely vertically, so we define a customized computation of cylinder `directions`:

```
In [5]: # Inherit from the M3C2 algorithm class to define a custom direction algorithm
        class M3C2_Vertical(py4dgeo.M3C2):
            def directions(self):
                return np.array([0, 0, 1]) # vertical vector orientation

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

```
# specify M3C2 parameters for our custom algorithm class
analysis.m3c2 = M3C2_Vertical(cyl_radii=(1.0,), max_distance=10.0, registration_err
```

```
[2023-03-29 15:54:54][INFO] Initializing Epoch object from given point cloud
[2023-03-29 15:54:54][INFO] Building KDTree structure with leaf parameter 10
[2023-03-29 15:54:54][INFO] Saving epoch to file 'C:\Users\k53\AppData\Local\Temp\tm
pfv704kty\corepoints.zip'
```

Now we add all the other epochs with their timestamps:

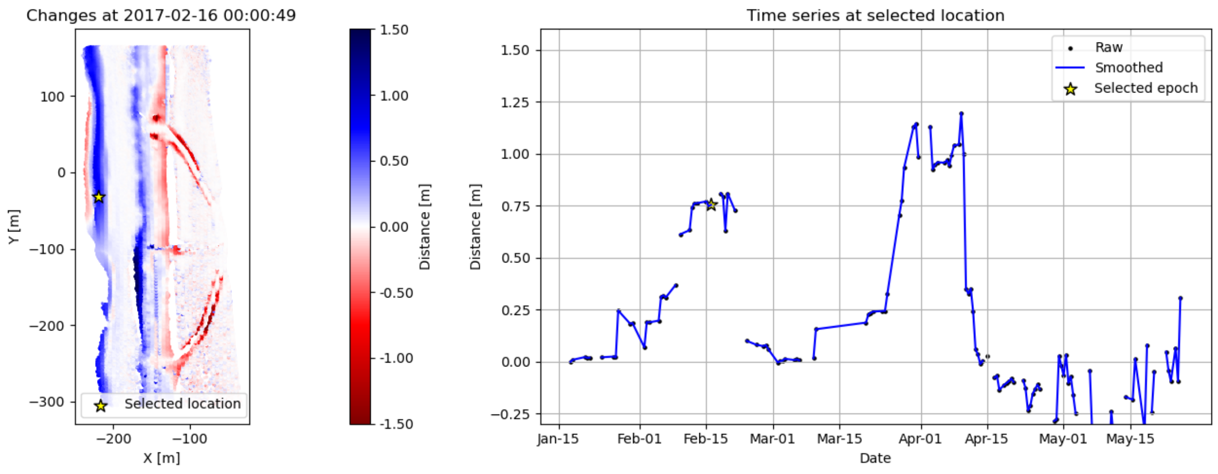
In [ ]:
```
# create a list to collect epoch objects
epochs = []
for e, pc_file in enumerate(pc_list[1:]):
    epoch_file = os.path.join(pc_dir, pc_file)
    epoch = py4dgeo.read_from_las(epoch_file)
    epoch.timestamp = timestamps[e]
    epochs.append(epoch)

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

# print the spatiotemporal analysis data for 3 corepoints and 5 epochs, respectivel
print(f"Space-time distance array:\n{analysis.distances[:3,:5]}")
print(f"Uncertainties of M3C2 distance calculation:\n{analysis.uncertainties['lodet
print(f"Timestamp deltas of analysis:\n{analysis.timedeltas[:5]}")
```

We visualize the changes in the scene for a selected epoch, together with the time series of surface changes at a selected location. The location here was selected separately in CloudCompare (as the corepoint id).

In [7]:
```
cp_idx_sel = 15162 # selected core point index
epoch_idx_sel = 28 # selected epoch index

# import plotting module
import matplotlib.pyplot as plt

# allow interactive rotation in notebook
%matplotlib inline

# create the figure
fig=plt.figure(figsize=(15,5))
ax1=fig.add_subplot(1,2,1)
ax2=fig.add_subplot(1,2,2)

# get the corepoints
corepoints = analysis.corepoints.cloud

# get change values of last epoch for all corepoints
distances = analysis.distances
distances_epoch = [d[epoch_idx_sel] for d in distances]

# get the time series of changes at a specific core point locations
coord_sel = analysis.corepoints.cloud[cp_idx_sel]
timeseries_sel = distances[cp_idx_sel]
```

```
# get the list of timestamps from the reference epoch timestamp and timedeltas
timestamps = [t + analysis.reference_epoch.timestamp for t in analysis.timedeltas]

# plot the scene
d = ax1.scatter(corepoints[:,0], corepoints[:,1], c=distances_epoch[:], cmap='seism
plt.colorbar(d, format=('%.2f'), label='Distance [m]', ax=ax1, pad=.15)

# add the location of the selected coordinate
ax1.scatter(coord_sel[0], coord_sel[1], facecolor='yellow', edgecolor='black', s=10
ax1.legend()

# configure the plot layout
ax1.set_xlabel('X [m]')
ax1.set_ylabel('Y [m]')
ax1.set_aspect('equal')
ax1.set_title('Changes at %s' % (analysis.reference_epoch.timestamp+analysis.timede

# plot the time series
ax2.scatter(timestamps, timeseries_sel, s=5, color='black', label='Raw')
ax2.plot(timestamps, timeseries_sel, color='blue', label='Smoothed')
ax2.set_xlabel('Date')

# add the epoch of the plotted scene
ax2.scatter(timestamps[epoch_idx_sel], timeseries_sel[epoch_idx_sel], facecolor='ye
ax2.legend()

# format the date labels
import matplotlib.dates as mdates
dtFmt = mdates.DateFormatter('%b-%d')
plt.gca().xaxis.set_major_formatter(dtFmt)

# configure the plot layout
ax2.set_ylabel('Distance [m]')
ax2.grid()
ax2.set_ylim(-0.3,1.6)
ax2.set_title('Time series at selected location')

plt.tight_layout()
plt.show()
```

```
[2023-03-29 15:56:48][INFO] Restoring epoch from file 'C:\Users\k53\AppData\Local\Te
mp\tmpr9igr1u0\reference_epoch.zip'
```



The map of changes in the scene shows us linear structures of sand deposition near the coast which can be interpreted as sand bars (with knowledge about coastal processes). This is confirmed by the surface behavior over time, expressed in the time series plot. However, the time series is quite noisy especially in this part of the beach, which is regularly covered by water during high tides (leading to missing data) and also varies strongly in surface moisture (influencing the LiDAR range measurement and causing noise). We therefore continue with temporal smoothing of the time series.

Temporal smoothing

You are dealing with a temporal subset of the original hourly time series. The effect of temporal measurement variability may therefore be less pronounced (compared to the assessment in, e.g., Anders et al., 2019). Nonetheless, you may apply temporal smoothing to reduce the influence of noise on your change analysis using a rolling median averaging of one week. This will also fill possible gaps in your data, e.g., lower ranges during poor atmospheric conditions or no data due to water coverage during tides on the lower beach part.

Visualize the raw and smoothed change values in the time series of your selected location.

We apply a rolling median with a defined temporal window of 14 epochs (corresponding to one week of 12-hourly point clouds) using the `temporal_averaging()` function in py4dgeo.

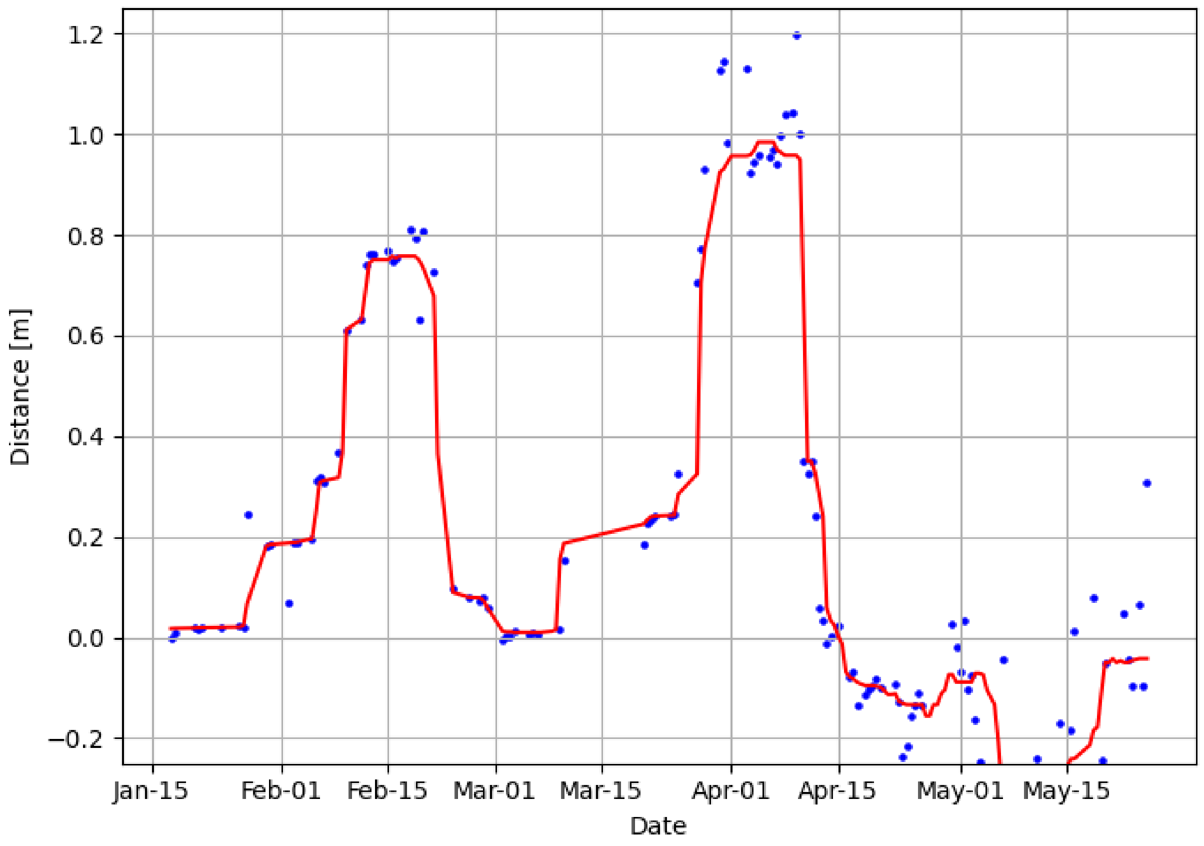
In [8]:
```
analysis.smoothed_distances = py4dgeo.temporal_averaging(analysis.distances, smooth
```

```
[2023-03-29 15:56:57][INFO] Starting: Smoothing temporal data
h:\conda_envs\etrainee\lib\site-packages\numpy\lib\nanfunctions.py:1217: RuntimeWarn
ing: All-NaN slice encountered
  return function_base._ureduce(a, func=_nanmedian, keepdims=keepdims,
[2023-03-29 15:57:20][INFO] Finished in 23.2358s: Smoothing temporal data
```

Now we can compare the raw and smoothed time series at our selected location:

In [9]:
```
# create the figure
fig, ax = plt.subplots(1,1,figsize=(7,5))

# plot the raw time series
ax.scatter(timestamps, timeseries_sel, color='blue', label='raw', s=5)

# plot the smoothed time series
timeseries_sel_smooth = analysis.smoothed_distances[cp_idx_sel]
ax.plot(timestamps, timeseries_sel_smooth, color='red', label='smooth')

# format the date labels
import matplotlib.dates as mdates
dtFmt = mdates.DateFormatter('%b-%d')
plt.gca().xaxis.set_major_formatter(dtFmt)
```

```
# add plot elements
ax.set_xlabel('Date')
ax.set_ylabel('Distance [m]')
ax.grid()
ax.set_ylim(-.25,1.25)

plt.tight_layout()
plt.show()
```



From the smoothed time series at the selected location, we can now much better interpret the surface behavior. In fact, we can distinctly observe that there were two consecutive occurrences of temporary deposition of several weeks. These represent two phases where sand bars are present. They can be extracted as individual objects by the 4D objects-by-change method. Before, we continue with time series clustering and the assessment of overall change patterns in the following.

Time series clustering

To derive characteristic change patterns on the sandy beach, perform k-means clustering of the time series following Kuschnerus et al. (2021). Assess the clustering for different selections of `k` numbers of clusters.

- Can you interpret the characteristics of different parts on the beach? Visualize example time series for different clusters.

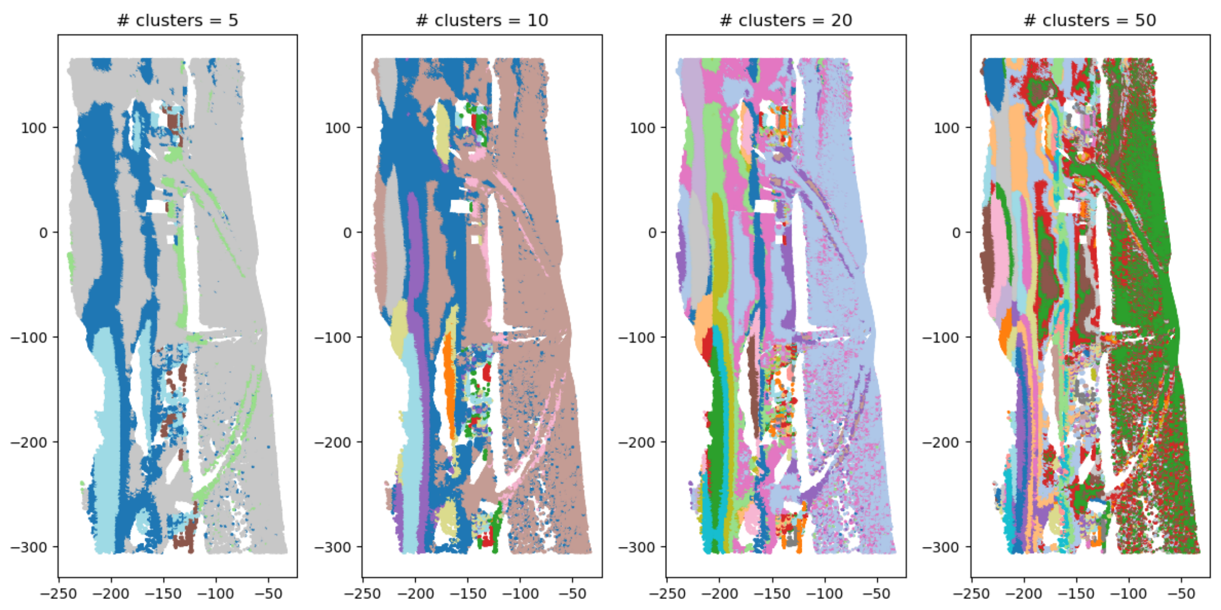
- From which number of clusters do you see a clear separation in overall units of the beach area?
- What are some detail change patterns that become visible for a higher number of clusters?

We perform k-means clustering for a set of `k` s at once and collect the resulting labels for subsequent assessment:

```
In [ ]:
# import kmeans clustering module from scikit-learn
from sklearn.cluster import KMeans

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

# define the number of clusters
ks = [5,10,20,50]

# create an array to store the labels
labels = np.full((distances.shape[0], len(ks)), np.nan)

# perform clustering for each number of clusters
for kidx, k in enumerate(ks):
    print(f'Performing clustering with k={k}...')
    nan_indicator = np.logical_not(np.isnan(np.sum(distances, axis=1)))
    kmeans = KMeans(n_clusters=k, random_state=0).fit(distances[nan_indicator, :])
    labels[nan_indicator, kidx] = kmeans.labels_
```

Now we can visualize the resulting change patterns for different numbers of clusters:

```
In [11]:
fig, axs = plt.subplots(1,4, figsize=(12,7))
(ax1,ax2,ax3,ax4) = axs

cmap_clustering = 'tab20'
sc1 = ax1.scatter(corepoints[:,0],corepoints[:,1],c=labels[:,0],cmap=cmap_clusterin

sc2 = ax2.scatter(corepoints[:,0],corepoints[:,1],c=labels[:,1],cmap=cmap_clusterin

sc3 = ax3.scatter(corepoints[:,0],corepoints[:,1],c=labels[:,2],cmap=cmap_clusterin

sc4 = ax4.scatter(corepoints[:,0],corepoints[:,1],c=labels[:,3],cmap=cmap_clusterin

ax_c = 0
for ax in axs:
    ax.set_aspect('equal')
    ax.set_title(f'# clusters = {ks[ax_c]}')
    ax_c+=1

plt.tight_layout()
plt.show()
```



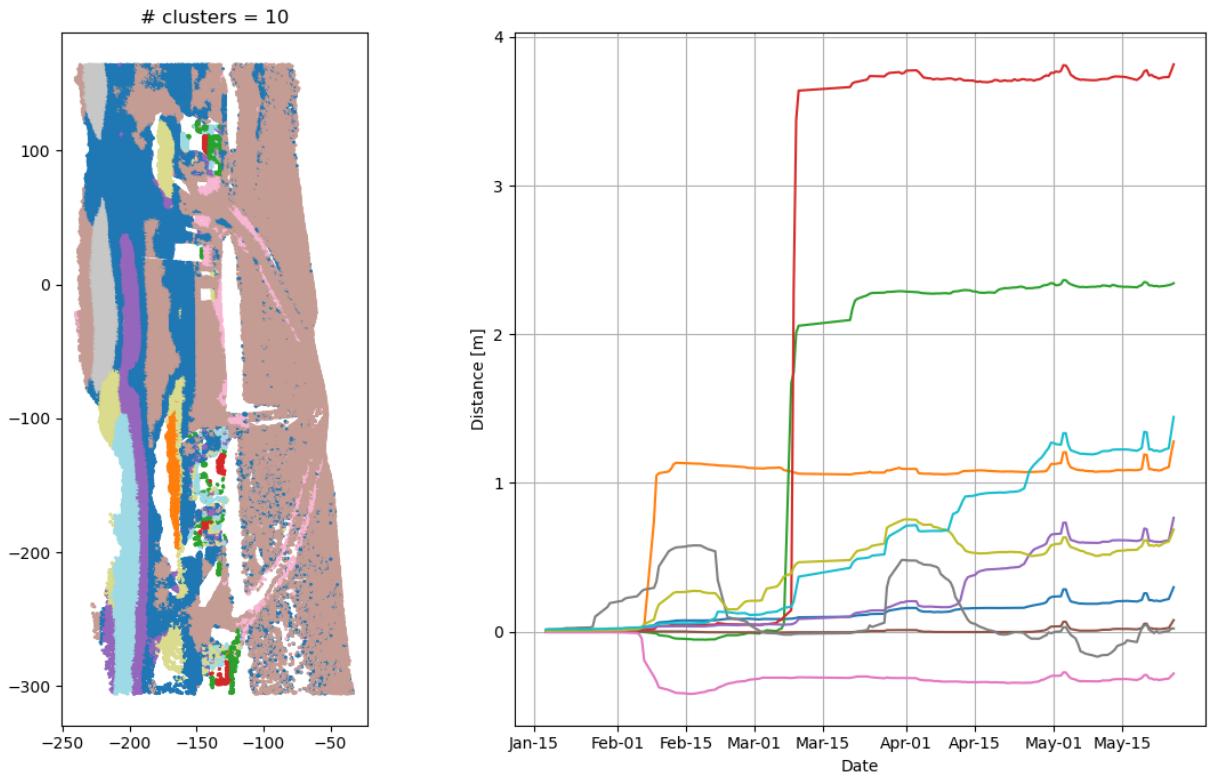
When using a small number of clusters (k=5), a large part of the beach is assigned to one cluster of assumingly little activity (gray area). From our exploration of changes in the scene at a selected epoch above, we further obtain two clusters with mainly deposition (blue clusters) and one cluster in the erosion areas around the pathway through the dunes. With a higher number of clusters (k=10 to k=50), the coarser clusters are further split up into (assumingly) finer patterns.

We can look into the time series properties within selected clusters, to interpret the change pattern they are representing. Here, we will check the clusters derived with k=10 by plotting the median values of all time series per cluster:

In [12]:
```
# create the figure
fig, axs = plt.subplots(1,2, figsize=(12,7))
ax1,ax2 = axs

# get the labels for the selected number of clusters
labels_k = labels[:,1]

# plot the map of clusters
sc = ax1.scatter(corepoints[:,0],corepoints[:,1],c=labels_k,cmap=cmap_clustering,s=

# plot the time series representing the clusters (median of each cluster)
for l in np.unique(labels_k):
    ax2.plot(timestamps, np.nanmedian(distances[labels_k==l,:], axis=0), label=f'cl

# format the date labels
import matplotlib.dates as mdates
dtFmt = mdates.DateFormatter('%b-%d')
plt.gca().xaxis.set_major_formatter(dtFmt)

# add plot elements
ax1.set_aspect('equal')
ax1.set_title(f'# clusters = {ks[1]}')
ax2.set_xlabel('Date')
```

```
ax2.set_ylabel('Distance [m]')
ax2.grid()

plt.tight_layout()
plt.show()
```

```
h:\conda_envs\etrainee\lib\site-packages\numpy\lib\nanfunctions.py:1215: RuntimeWarn
ing: Mean of empty slice
  return np.nanmean(a, axis, out=out, keepdims=keepdims)
```



Now we can relate the temporal behavior to the spatial clusters. For example, we see strong and sudden surface increases for the orange and red clusters (with different timing and magnitude). The gray cluster (occuring in two spatial extents) represents sand bars, which we know from our time series example above. The large coverage of the brown cluster shows a time series with little activity - except for fluctuations, especially towards the end of the observation period, which is contained in all clusters. We can assume that this represents some measurement effects in the data that could not be corrected by the preprocessing and alignment procedure ([Vos et al., 2022]).

Extraction of 4D objects-by-change

Now use the 4D objects-by-change (4D-OBC) method to identify individual surface activities in the beach scene. The objective is to extract temporary occurrences of accumulation or erosion, as occurs when a sand bar is formed during some timespan, or when sand is deposited by anthropogenic works. This type of surface activity is implemented with the original seed detection in py4dgeo, so you do not need to customize the algorithm. Decide for suitable parameters following Anders et al. (2021) - but bear in mind that we are using a different temporal resolution, so you may need to adjust the temporal change detection.

Perform the extraction for selected seed locations, e.g. considering interesting clusters of change patterns identified in the previous step. In principle, the spatiotemporal segmentation can also be applied to the full dataset (all time series at all core point locations are used as potential seed candidates), but long processing time needs to be expected.

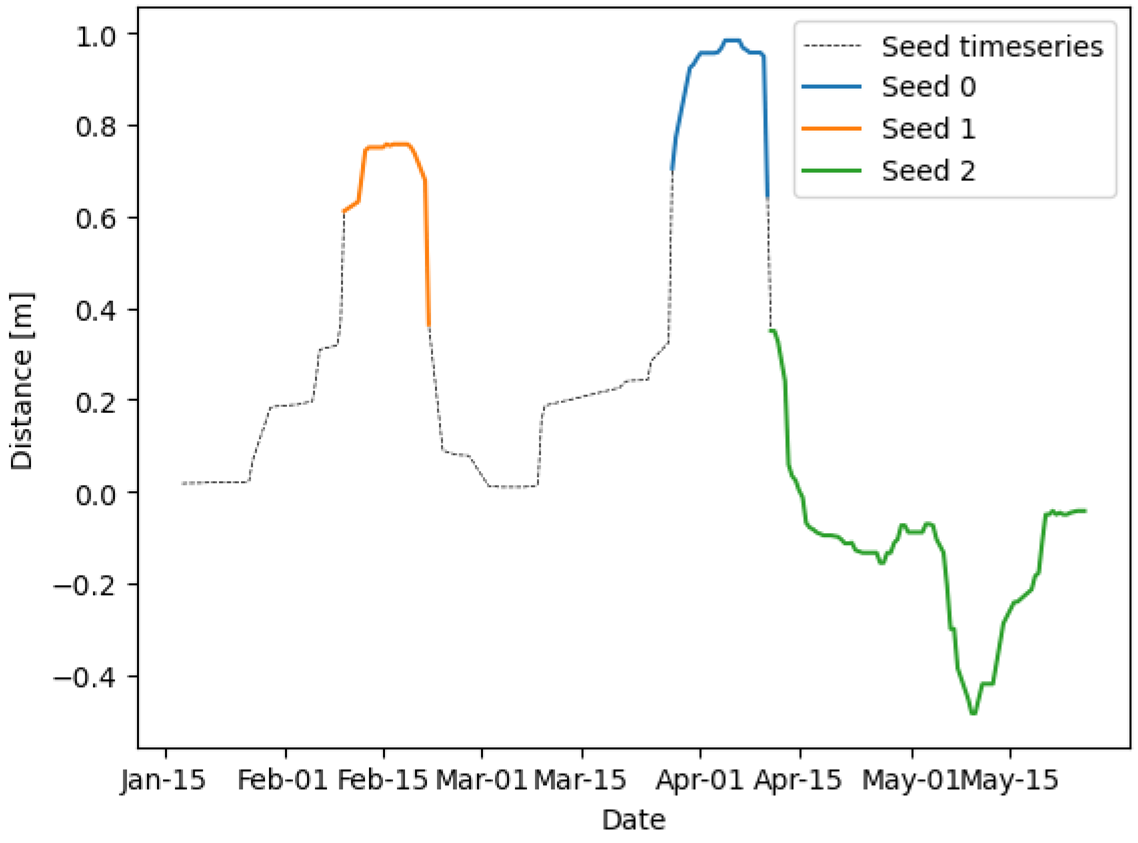
In this solution, we will use the selected location from above to extract the sand bars as 4D object-by-change. The strength of the method is that the occurrences are identified individually, even though they have spatial overlap, as they can be separated in the time series information.

We instantiate the `RegionGrowingAlgorithm` class using the parameters of Anders et al, 2021, and run the object extraction:

In [13]:
```
# 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=[cp_idx_sel])

# run the algorithm
analysis.invalidate_results(seeds=True, objects=True, smoothed_distances=False) # o
objects = algo.run(analysis)
```

```
[2023-03-29 15:58:32][INFO] Removing intermediate results from the analysis file
I:\etrainee_data\kijkduin/kijkduin.zip
[2023-03-29 15:58:32][INFO] Starting: Find seed candidates in time series
[2023-03-29 15:58:32][INFO] Finished in 0.0017s: Find seed candidates in time series
[2023-03-29 15:58:32][INFO] Starting: Sort seed candidates by priority
[2023-03-29 15:58:32][INFO] Finished in 0.0016s: Sort seed candidates by priority
[2023-03-29 15:58:32][INFO] Starting: Performing region growing on seed candidate 1/
3
[2023-03-29 15:58:32][INFO] Finished in 0.0565s: Performing region growing on seed c
andidate 1/3
[2023-03-29 15:58:32][INFO] Starting: Performing region growing on seed candidate 2/
3
[2023-03-29 15:58:32][INFO] Finished in 0.0879s: Performing region growing on seed c
andidate 2/3
[2023-03-29 15:58:32][INFO] Starting: Performing region growing on seed candidate 3/
3
[2023-03-29 15:58:33][INFO] Finished in 0.5324s: Performing region growing on seed c
andidate 3/3
```

In [14]:
```
seed_timeseries = analysis.smoothed_distances[cp_idx_sel]
plt.plot(timestamps,seed_timeseries, c='black', linestyle='--', linewidth=0.5, labe
for sid, example_seed in enumerate(analysis.seeds):
    seed_end = example_seed.end_epoch
    seed_start = example_seed.start_epoch
    seed_cp_idx = example_seed.index

    plt.plot(timestamps[seed_start:seed_end+1], seed_timeseries[seed_start:seed_end

# format the date labels
dtFmt = mdates.DateFormatter('%b-%d')
plt.gca().xaxis.set_major_formatter(dtFmt)

# add plot elements
plt.xlabel('Date')
plt.ylabel('Distance [m]')

plt.legend()
plt.show()
```



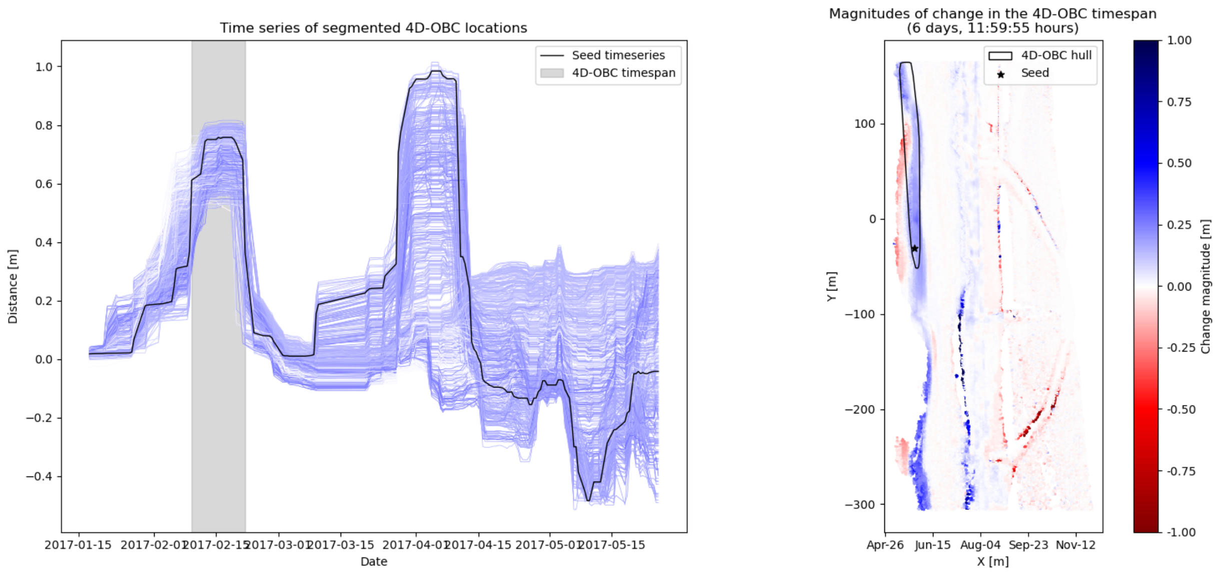
At the selected location, three separate surface activities are detected as seed for 4D-OBC extraction. We may not be satisfied with the determination of the timing. In a real analysis, we would now aim to improve the temporal detection - either by using a different approach of seed detection (cf. algorithm customization), or by postprocessing the temporal segments from the current seed detection.

Here, we use the result and look into the spatial object properties:

In [15]:
```
fig, axs = plt.subplots(1,2, figsize=(15,7))
ax1,ax2 = axs
```

```
# get indices of 4D-OBC
sel_seed_idx = 1
idxs = objects[sel_seed_idx].indices

# get change values at end of object for each location
epoch_of_interest = int((objects[sel_seed_idx].end_epoch - objects[sel_seed_idx].st
distances_of_interest = analysis.smoothed_distances[:, epoch_of_interest]

# get the change magnitude between end and start of object for each location
magnitudes_of_interest = analysis.smoothed_distances[:, epoch_of_interest] - analys

# set the colormap according to magnitude at each location in the object
crange = 1.0
import matplotlib.colors as mcolors
cmap = plt.get_cmap('seismic_r').copy()
norm = mcolors.CenteredNorm(halfrange=crange)
cmapvals = norm(magnitudes_of_interest)

# plot the timeseries of the segmented locations (colored by time series similarity
for idx in idxs[::10]:
    ax1.plot(timestamps, analysis.smoothed_distances[idx], c=cmap(cmapvals[idx]), l

# plot the seed time series
ax1.plot(timestamps, analysis.smoothed_distances[cp_idx_sel], c='black', linewidth=

# fill the area of the object
ax1.axvspan(timestamps[objects[sel_seed_idx].start_epoch], timestamps[objects[sel_s

# add legend and format the date labels
dtFmt = mdates.DateFormatter('%b-%d')
plt.gca().xaxis.set_major_formatter(dtFmt)
ax1.legend()

# get subset of core points incorporated in 4D-OBC
cloud = analysis.corepoints.cloud
subset_cloud = cloud[idxs,:2]

# plot coordinates colored by change values at end magnitude of object
d = ax2.scatter(cloud[:,0], cloud[:,1], c = magnitudes_of_interest, cmap='seismic_r
plt.colorbar(d, format=('%.2f'), label='Change magnitude [m]', ax=ax2)
ax2.set_aspect('equal')

# plot convex hull of 4D-OBC
from scipy.spatial import ConvexHull
from matplotlib.patches import Polygon
hull = ConvexHull(subset_cloud)
ax2.add_patch(Polygon(subset_cloud[hull.vertices,0:2], label = '4D-OBC hull', fill

# plot seed location of 4D-OBC
ax2.scatter(cloud[cp_idx_sel,0], cloud[cp_idx_sel,1], marker = '*', c = 'black', la

# add plot elements
ax1.set_title('Time series of segmented 4D-OBC locations')
ax1.set_xlabel('Date')
ax1.set_ylabel('Distance [m]')
```

```
ax2.set_title(f'Magnitudes of change in the 4D-OBC timespan\n({timestamps[epoch_of_
ax2.set_xlabel('X [m]')
ax2.set_ylabel('Y [m]')
ax2.legend(loc='upper right')

plt.tight_layout()
plt.show()
```



You may now check the different spatial extents for the three objects extracted from this seed location. In subsequent analysis, the spatial-temporal extent of each object can be used to describe individual activities (e.g., their change rates, magnitudes, ...) and relating them to one another in their timing and spatial distribution.

We will not further go into the analysis here, but note that the 4D point clouds at this sandy beach are features as a research-oriented case study!

References

- Anders, K., Lindenbergh, R. C., Vos, S. E., Mara, H., de Vries, S., & Höfle, B. (2019). High-Frequency 3D Geomorphic Observation Using Hourly Terrestrial Laser Scanning Data Of A Sandy Beach. ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., IV-2/W5, pp. 317-324. doi: 10.5194/isprs-annals-IV-2-W5-317-2019.
- Anders, K., Winiwarter, L., Mara, H., Lindenbergh, R., Vos, S. E., & Höfle, B. (2021). Fully automatic spatiotemporal segmentation of 3D LiDAR time series for the extraction of natural surface changes. ISPRS Journal of Photogrammetry and Remote Sensing, 173, pp. 297-308. doi: 10.1016/j.isprsjprs.2021.01.015.
- Kuschnerus, M., Lindenbergh, R., & Vos, S. (2021). Coastal change patterns from time series clustering of permanent laser scan data. Earth Surface Dynamics, 9 (1), pp. 89-103. doi: 10.5194/esurf-9-89-2021.

- Vos, S., Anders, K., Kuschnerus, M., Lindenberg, R., Höfle, B., Aarnikhof, S. & Vries, S. (2022). A high-resolution 4D terrestrial laser scan dataset of the Kijkduin beach-dune system, The Netherlands. Scientific Data, 9:191. doi: 10.1038/s41597-022-01291-9.