# pyIDI Documentation

***Release 0.17***

**Klemen Zaletelj, Domen Gorjup, Janko Slavič**

**Sep 06, 2021**

# Contents:

CHAPTER 1

# pyIDI

`pyidi` is a python package for displacement identification from raw video.

Currently the pyIDI method works with Photron `.cih` and `.cihx` files, however, `numpy.ndarray` can also be passed as `cih_file` argument. If an array is passed, it must have a shape of: `(n time points, image height, image width)`.

The basic usage of the package is presented.

## 1.1 Loading the video

First create the `pyIDI` object:

```
video = pyidi.pyIDI('filename.cih')
```

## 1.2 Setting the points

Displacements are computed for certain points or certain regions of interest that are represented by a point.

Points must be of shape `n_points x 2`:

```
points = [[1, 2],
          [1, 5],
          [2, 10]]
```

where the first column indicates indices along **axis 0**, and the second column indices along **axis 1**.

The points must be passed to `pyIDI` object:

```
video.set_points(points=points)
```

If the points are not known, a *Point selection UI* or newer *Napari image viewer* can be used to select the points.

## 1.3 Setting the method

The method for displacement identification must be selected:

```
video.set_method(method='sof') # Simplified optical flow method
```

After the method is selected, the arguments can be configured. Note that the docstring is now showing the required arguments for the selected method.

```
video.method.configure(*args, **kwargs)
```

For more details on the available methods, see the currently implemented *Displacement identification methods*.

## 1.4 Get displacement

Finally, displacements can be identified:

```
displacements = video.get_displacements()
```

## 1.5 Saved analysis

The settings of the analysis and the identified displacements are saved in a directory next to the loaded `cih_file`.

Directory content before the analysis:

- video_to_analyze.cih

Directory content after the analysis:

- video_to_analyze.cih
- video_to_analyze_pyidi_analysis
  - analysis_001
    - points.pkl
    - results.pkl
    - settings.txt

## 1.6 Loading saved analysis

The saved analysis can be loaded using the `load_analysis` function:

```
analysis_path = 'video_to_analyze_pyidi_analysis/analysis_001'

video_loaded, info_dict = pyidi.load_analysis(analysis_path)
```

Now we can access the `video_loaded` attributes, e.g.:

```
video_loaded.displacements
```

CHAPTER 2

# Point selection UI

A convinient UI is available to make the point selection easier.

To use the UI, the `pyIDI` object must first be available. It is created by:

```
video = pyidi.pyIDI(cih_file)
```

A `ROISelect` object can then be created:

```
Points = pyidi.selection.ROISelect(video, roi_size=(21, 21), noverlap=0)
```

where `roi_size` is the size of a single Region-Of-Interest/subset in `y` and `x` direction respsectivly. The `noverlap` argument prescribes the overlap of the neighbouring ROIs. The density of the grid can be adjusted using `noverlap`.

The UI enables multiple modes of point selection. Currently, the following are supported:

- `ROI grid`: A regular grid of ROIs is created based on the selected polygon.
- `Deselect ROI polygon`: After defining the polygon and getting the points, this method can be used to define a polygon within which the points are not selected.
- `Only polygon`: Same as ROI grid but the points are not computed. Only polygon points are available.
- `Manual ROI select`: Manually select the ROIs at desired locations.

Once the selection in the UI is complete, the points can be retrieved:

```
points = Points.points
```

CHAPTER 3

# Napari image viewer

Interactive image viewer **napari** is implemented and can be used for viewing video and selecting points. More information about napari can be obtained here.

## 3.1 Basic usage

To use the image viewer, the `pyIDI` object must first be available. It is created by:

```
video = pyidi.pyIDI('cih_file')
```

Image viewer is launched by calling `pyIDI` object as a function:

```
video.gui()
```

If the `method` and `points` are already set in the `video` object, these will show in the interface. Otherwise, the user can set them in the UI.

Viewer is divided in few key areas. On the left side from main canvas there are: layer controls (editing layer colors, brightness), layer buttons, layer list and viewer buttons. On the right side there is a dock widget used for point selection. Slider below the canvas can be used for scrolling through video. By selecting image layer from layer list you can toggle layer visibility and adjust video colors or lightning.

## 3.2 Setting the method

## 3.3 Individual points selection

To select individual points, choose `Points` from layer list on the left side. Then choose `Add points` button from layer control or press `P` key. Select points by clicking on screen, use `space` to zoom and move across image. By choosing `Select points` button from layer control or pressing `S` key and selecting points on canvas, points

can be moved, deleted or scaled with `point size` slider. Points are added to `pyIDI` object in `video.points` by pressing `Set points` on the right side of UI.

## 3.4 Area selection

To select an area for creating a grid of points, choose `Area selection` from layer list. Then under layer controls choose `Add rectange` (`R` key) or `Add polygons` (`P` key). When drawing polygons, press `esc` key to stop adding points. An area inside of already selected area can be deselected using `Area deselection` layer from layer list. Shapes can be moved, scaled, edited and deleted using `Select vertices` (`D` key) and `Select shapes` (`S` key) buttons form layer controls. Grid parameters are set using dock on the right side from canvas. Grid is shown and points are added to `pyIDI` object after pressing `Set points`.

## 3.5 Configure

Once the points are set, the method can be configured by editing the settings shown in the UI. All of the changes done in the UI change the attributes of the `pyIDI` object.

## 3.6 Displacement computation

After the method is selected, the points are set and the configuration is performed, the displacements can be computed.

CHAPTER 4

# Displacement identification methods

## 4.1 Simplified Optical Flow (SOF)

Javh, J., Slavič, J., & Boltežar, M. (2017). The subpixel resolution of optical-flow-based modal analysis. Mechanical Systems and Signal Processing, 88, 89–99. https://doi.org/10.1016/j.ymssp.2016.11.009

## 4.2 Lucas-Kanade (LK)

Lucas, B. D., & Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. In Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2 (pp. 674–679). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1623264.1623280

CHAPTER 5

# Documenting the code

## 5.1 Requirements

- *Sphinx* (installed by default with anaconda)

```
pip install sphinx
```

### 5.1.1 Automatic code documentation with autodoc

The Sphinx autodoc extension automatically includes our Python modules documentation in the generated Sphinx documentation.

## 5.2 Addind a new module documentaiton to the build

It is most likely not practical to iclude all of our apps Python modules in the build. Only the modules and classes with the most extensive documentation (docstrings), and the ones where most developer updates are expected should be included.

To add another module's documentation to the build, add a new entry to the `doc/code/modules.rst` file, with the correct relative Python path to the module. For example, the `views.py` documentation is included by:

```
Tools
-----
.. automodule:: pyidi.tools
    :members:
```

For more information, see the autodoc documentation.

## 5.3 Docstring style: reStructuredText

- default docstring style in PyCharm
- `"autoDocstring.docstringFormat": "sphinx"` in VSCode Python Docstring extenion

Example:

```
def function1(self, arg1, arg2, arg3):
"""returns (arg1 / arg2) + arg3

This is a longer explanation, which may include math with latex syntax
:math:`\\alpha`.
Then, you need to provide optional subsection in this order (just to be
consistent and have a uniform documentation. Nothing prevent you to
switch the order):

  - parameters using ``:param <name>: <description>``
  - type of the parameters ``:type <name>: <description>``
  - returns using ``:returns: <description>``
  - examples (doctest)
  - seealso using ``.. seealso:: text``
  - notes using ``.. note:: text``
  - warning using ``.. warning:: text``
  - todo ``.. todo:: text``

**Advantages**:
 - Uses sphinx markups, which will certainly be improved in future
   version
 - Nice HTML output with the See Also, Note, Warnings directives


**Drawbacks**:
 - Just looking at the docstring, the parameter, type and  return
   sections do not appear nicely

:param arg1: the first value
:param arg2: the first value
:param arg3: the first value
:type arg1: int, float,...
:type arg2: int, float,...
:type arg3: int, float,...
:returns: arg1/arg2 +arg3
:rtype: int, float

:Example:

>>> import template
>>> a = template.MainClass1()
>>> a.function1(1,1,1)
2

.. note:: can be useful to emphasize
    important feature
.. seealso:: :class:`MainClass2`
.. warning:: arg2 must be non-zero.
.. todo:: check that arg2 is non zero.
"""
return arg1/arg2 + arg3
```

# CHAPTER 6

# pyIDI source code

## 6.1 pyIDI base class

**class** pyidi.pyidi.**pyIDI**(*cih_file*)
The pyIDI base class represents the video to be analysed.

**close_video**()
Close the .mraw video memmap.

**get_displacements**(*autosave=True*, ***kwargs*)
Calculate the displacements based on chosen method.

Method docstring: — Method is not set. Please use the *set_method* method. —

**set_method**(*method*, ***kwargs*)
Set displacement identification method on video. To configure the method, use *method.configure()*

Available methods: — [Available method names and descriptions go here.] —

**Parameters** **method** (*IDIMethod or str*) – the method to be used for displacement identification.

**set_points**(*points=None*, *method=None*, ***kwargs*)
Set points that will be used to calculate displacements. If *points* is None and a *method* has aready been set on this *pyIDI* instance, the *method* object's *get_point* is used to get method-appropriate points.

**show_field**(*field*, *scale=1.0*, *width=0.5*)
Show displacement field on image.

**Parameters**

- **field** (*ndarray*) – Field of displacements (number_of_points, 2)
- **scale** – scale the field, defaults to 1.
- **scale** – float, optional
- **width** – width of the arrow, defaults to 0.5
- **width** – float, optional

**show_points**(***kwargs*)
Show selected points on image.

## 6.2 Displacement identification methods

### 6.2.1 IDIMetod base class

**class** pyidi.methods.idi_method.**IDIMethod**(*video*, **args*, ***kwargs*)

Common functions for all methods.

**calculate_displacements**(*video*, **args*, ***kwargs*)

Calculate the displacements of set points here. The result should be saved into the *self.displacements* attribute.

**configure**(**args*, ***kwargs*)

Configure the displacement identification method here.

### 6.2.2 Simplified optical flow

**class** pyidi.methods._simplified_optical_flow.**PickPoints**(*video*, *subset*, *axis*, *min_grad*)

Pick the area of interest.

Select the points with highest gradient in vertical direction.

**inside_polygon**(*x*, *y*, *points*)

Return True if a coordinate (x, y) is inside a polygon defined by a list of verticies [(x1, y1), (x2, x2), ... , (xN, yN)].

Reference: http://www.ariel.com.au/a/python-point-int-poly.html

**class** pyidi.methods._simplified_optical_flow.**SimplifiedOpticalFlow**(*video*, **args*, ***kwargs*)

Displacmenet computation based on Simplified Optical Flow method [1].

**Literature:**

**[1] Javh, J., Slavič, J., & Boltežar, M. (2017). The subpixel resolution** of optical-flow-based modal analysis. Mechanical Systems and Signal Processing, 88, 89–99.

**[2] Lucas, B. D., & Kanade, T. (1981). An Iterative Image Registration** Technique with an Application to Stereo Vision. In Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2 (pp. 674–679). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

**calculate_displacements**(*video*)

Calculate the displacements of set points here. The result should be saved into the *self.displacements* attribute.

**configure**(*subset_size=3*, *pixel_shift=False*, *convert_from_px=1.0*, *mraw_range='all'*, *mean_n_neighbours=0*, *zero_shift=False*, *progress_bar=True*, *reference_range=(0, 100)*)

Set the attributes, compute reference image and gradients.

**Parameters**

- **video** (*object*) – 'parent' object
- **subset_size** – size of the averaging subset, defaults to 3
- **subset_size** – int, optional
- **pixel_shift** – use pixel shift or not?, defaults to False
- **pixel_shift** – bool, optional
- **convert_from_px** – distance unit per pixel, defaults to 1.

- **convert_from_px** – float or int, optional
- **mraw_range** – what range of images to calculate into displacements, defaults to 'all'
- **mraw_range** – str or tuple, optional
- **mean_n_neighbours** – average the displacements of neighbouring points (how many points), defaults to 0
- **mean_n_neighbours** – int, optional
- **zero_shift** – shift the mean of the signal to zero?, defaults to False
- **zero_shift** – bool, optional
- **progress_bar** – show progress bar while calculating the displacements, defaults to True
- **progress_bar** – bool, optional
- **reference_range** – what range of images is averaged into reference image, defaults to (0, 100)
- **reference_range** – tuple, optional

**displacement_averaging**()
Calculate the average of displacements.

**static get_points**(*video*, ***kwargs*)
Determine the points.

**pixel_shift**()
Pixel shifting implementation.

**reference**(*images*, *subset_size*)
Calculation of the reference image, image gradients and gradient amplitudes.

**Parameters**

- **images** – Images to average. Usually the first 100 images.
- **subset_size** – Size of the subset to average.

**Returns** Reference image, image gradient in 0 direction, image gradient in 1 direction, gradient magnitude

**subset**(*data*, *subset_size*)
Calculating a filtered image.

Calculates a filtered image with subset of d. It sums the area of d x d.

**Parameters**

- **data** – Image that is to be filtered.
- **subset_size** – Size of the subset.

**Returns** Filtered image.

### 6.2.3 The Lucas-Kanade algorithm for translations

**class** pyidi.methods._lucas_kanade.**LucasKanade**(*video*, **args*, ***kwargs*)
Translation identification based on the Lucas-Kanade method using least-squares iterative optimization with the Zero Normalized Cross Correlation optimization criterium.

**calculate_displacements**(*video*, ***kwargs*)
Calculate displacements for set points and roi size.

kwargs are passed to *configure* method. Pre-set arguments (using configure) are NOT changed!

**clear_temp_files**()
Clearing the temporary files.

**configure**(*roi_size=(9, 9)*, *pad=2*, *max_nfev=20*, *tol=1e-08*, *int_order=3*, *verbose=1*, *show_pbar=True*, *processes=1*, *pbar_type='atpbar'*, *multi_type='mantichora'*, *resume_analysis=True*, *process_number=0*, *reference_image=0*, *mraw_range='full'*, *use_numba=False*)
Displacement identification based on Lucas-Kanade method, using iterative least squares optimization of translatory transformation parameters to determine image ROI translations.

**Parameters**

- **video** (*object*) – parent object
- **roi_size** (*tuple, list, optional*) – (h, w) height and width of the region of interest. ROI dimensions should be odd numbers. Defaults to (9, 9)
- **pad** (*int, optional*) – size of padding around the region of interest in px, defaults to 2
- **max_nfev** (*int, optional*) – maximum number of iterations in least-squares optimization, defaults to 20
- **tol** (*float, optional*) – tolerance for termination of the iterative optimization loop. The minimum value of the optimization parameter vector norm.
- **int_order** (*int, optional*) – interpolation spline order
- **verbose** (*int, optional*) – show text while running, defaults to 1
- **show_pbar** (*bool, optional*) – show progress bar, defaults to True
- **processes** (*int, optional, defaults to 1.*) – number of processes to run
- **pbar_type** (*str, optional*) – type of the progress bar ('tqdm' or 'atpbar'), defaults to 'atpbar'
- **multi_type** (*str, optional*) – type of multiprocessing used ('multiprocessing' or 'mantichora'), defaults to 'mantichora'
- **resume_analysis** – if True, the last analysis results are loaded and computation continues from last computed time point.
- **process_number** (*int, optional*) – User should not change this (for multiprocessing purposes - to indicate the process number)
- **reference_image** (*int or tuple or ndarray*) – The reference image for computation. Can be index of a frame, tuple (slice) or numpy.ndarray that is taken as a reference.
- **mraw_range** (*tuple or "full"*) – Part of the video to process. If "full", a full video is processed. If first element of tuple is not 0, a appropriate reference image should be chosen.
- **use_numba** (*bool*) – Use numba.njit for computation speedup. Currently not implemented.

**create_settings_dict**()
Make a dictionary of the chosen settings.

**create_temp_files**(*init_multi=False*)
Temporary files to track the solving process.

This is done in case some error occures. In this eventuality the calculation can be resumed from the last computed time point.

**Parameters** **init_multi** (*bool, optional*) – when initialization multiprocessing, defaults to False

**optimize_translations**(*G*, *F_spline*, *maxiter*, *tol*, *d_subpixel_init=(0, 0)*)
Determine the optimal translation parameters to align the current image subset *G* with the interpolated reference image subset *F*.

**Parameters**

- **G** (array of shape *roi_size*) – the current image subset.
- **F_spline** (*scipy.interpolate.RectBivariateSpline*) – interpolated referencee image subset
- **maxiter** (*int*) – maximum number of iterations
- **tol** (*float*) – convergence criterium
- **d_subpixel_init** – initial subpixel displacement guess, relative to the integrer position of the image subset *G*

**Returns** the obtimal subpixel translation parameters of the current image, relative to the position of input subset *G*.

**Return type** array of size 2

**resume_temp_files**()
Reload the settings written in the temporary files.

When resuming the computation of displacement, the settings are loaded from the previously created temporary files.

**roi_size**
*roi_size* attribute getter

**show_points**(*video*, *figsize=(15, 5)*, *cmap='gray'*, *color='r'*)
Shoe points to be analyzed, together with ROI borders.

**Parameters**

- **figsize** – matplotlib figure size, defaults to (15, 5)
- **cmap** – matplotlib colormap, defaults to 'gray'
- **color** – marker and border color, defaults to 'r'

**temp_files_check**()
Checking the settings of computation.

The computation can only be resumed if all the settings and data are the same as with the original analysis. This function checks that (writing all the setting to dict and comparing the json dump of the dicts).

If the settings are the same but the points are not, a new analysis is also started. To set the same points, check the *temp_pyidi* folder.

**Returns** Whether to resume analysis or not

**Return type** bool

**update_log**(*last_time*)
Updating the log file.

A new last time is written in the log file in order to track the solution process.

**Parameters** **last_time** (*int*) – Last computed time point (index)

pyidi.methods._lucas_kanade.**multi**(*video*, *processes*)
Splitting the points to multiple processes and creating a pool of workers.

**Parameters**

- **video** (*object*) – the video object with defined attributes

- **processes** (*int*) – number of processes. If negative, the number of processes is set to *psutil.cpu_count* + *processes*.

**Returns** displacements

**Return type** ndarray

pyidi.methods._lucas_kanade.**worker**(*points*, *idi_kwargs*, *method_kwargs*, *i*)

A function that is called when for each job in multiprocessing.

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## p

# Index

## T



## U



## W