autopew Documentation

Release 0.1.2

Morgan Williams & Louise Schoneveld

Nov 13, 2020

Contents

autopew is designed for arbitrary translation between planar/2D Cartesian coordinate systems using affine transforms and human-in-the-loop workflows.

This is applied to integrating coordinate systems across analytical instrumentation, with each instrument typically having its own individual coordinate systems based on imagery and/or a sample stage. **autopew** also includes functions for importing and exporting files, for automated generation of point sets within a relevant format for each piece of analytical instrumentation. **autopew** outputs currently included a .scancsv file which can be directly imported into Chromium laser ablation navigation software.

CHAPTER 1

Why use **autopew**

autopew is designed for easy referencing between analytical equipment and/or images. This allows the time spend on analytical equipment to be more effectively used for data collection rather than spending valuable time locating the areas of interest.

This software also allows for tracking of the context of in-situ microanalysis by allowing reference to large images and areas which will allow for new insights into what effects chemistry of given particles with reference to their location and micro-environment. We can then track the analysis between different analytical equipment and make inferences on macroscale processes from well characterised in-situ microanalysis[1].

Although primarily designed for use of laser ablation analysis on geological material this software can be used for any microanalytical technique, including electron microprobe analysis, x-ray fluorescence mapping, scanning electron microscopy and ion beam analysis.

See also:

For outlined examples of how autopew is used, see Examples

[1] Pearce, M. A., Godel, B. M., Fisher, L. A., Schoneveld, L. E., Cleverly, J. S., Oliver, N. H. S., and Nugus, M. (2017). Microscale data to macroscale processes: a review of microcharacterization applied to mineral systems: Geological Society, London, Special Publications,, v. 453 doi:10.1144/SP453.3.

CHAPTER 2

What is **autopew** not?

- Not currently capable of 3D affine transforms (i.e. no 'focus' attribute).

The current development plan for **autopew** can be found here.

2.1 Installation

autopew is a private repo on GitHub. If you have access on GitHub, you can install it with pip (you may be prompted for user-password):

```
pip install git+https://github.com/morganjwilliams/autopew.git#egg=autopew
# or, for the develop version
pip install git+https://github.com/morganjwilliams/autopew.git@develop#egg=autopew
```

Alternatively, you can also clone it locally and install with pip:

```
pip install <repo-directory>
# e.g if you navigate to the directory, this is then:
pip install .
```

2.2 Getting Started

Note: This page is under construction. Feel free to send through suggestions or questions.

2.2.1 Set up

To use this software you need in install Python. Anaconda is a great way to install python and included popular software for editing and interacting with python script such as Spyder and Jupyter.

2.2.2 Installation

See also:

Installation

For in depth installation processes see the page above. For most purposes you should be able to install **autopew** by typing the following in a terminal ("Anaconda Prompt" application on your computer if you're using Anaconda)

```
pip install autopew
```

If you're already installed autopew and would like the most up-to-date version, type this into a terminal:

```
pip install --upgrade autopew
```

2.2.3 Writing and editing code

autopew is designed to be very beginner friendly, but if you're completely new to python consider visiting some free online courses about the basic Python concepts. Codecademy is a great jumping off point.

2.3 Examples

Here are some examples of what **autopew** can do and how you can do it. Before using these examples, please visit the installation and getting started pages.

See also:

Installation , Getting Started

2.3.1 Use Cases

Here we outline some common use cases as well as what is required for use. This includes translating from pixel coordinates on images to analytical stages and translating between analytical stages directly.

Use Cases

autopew is designed for easy referencing between analytical equipment and/or images. This allows users to easily transfer between techniques such as electron probe, laser ablation, scanning electron microscope or other imaging techniques. autopew will allow consistent measurements of the same grains via different techniques and give spatial context to chemical data.

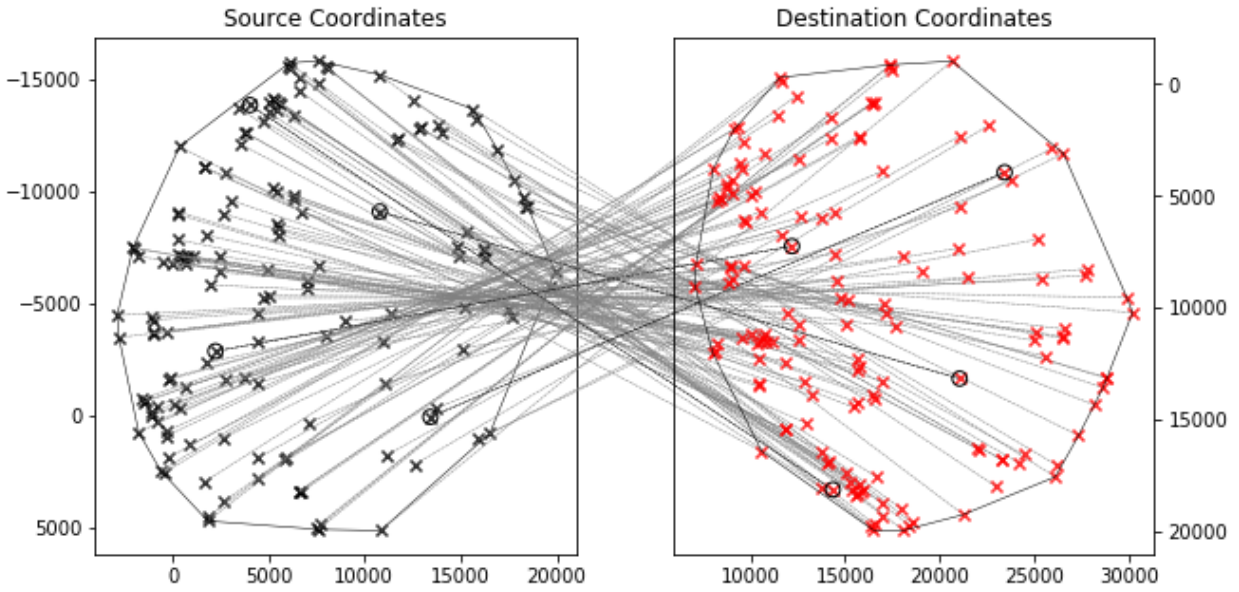
There are a number of use cases that autopew is suited for:

Two Sets of Points

E.g. coordinates from stages This case is for transfer between various stage coordinate system (e.g. electron probe and laser ablation) which can allow for measurement of the same grain with different techniques.

What you need:

- X,Y coordinates of the points you wish to analyse
- at least 3 points in the new coordinate system



autopew can translate points with rotation, and shear.

Image and Set of Points

If you have high resolution microscope images or images from other sources such as X-ray fluorescence mapping you can use pixel coordinates and convert the chosen points to the new stage coordinates.

This allows you to quickly pick phases for analysis without the need to waste analytical time locating and programming the coordinates of the phases.

What you need:

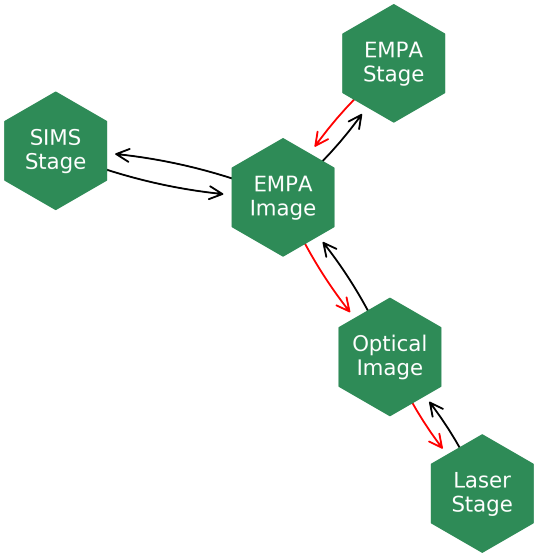
- an image of high enough resolution to identify the target phases
- 3 points in the new coordinate system that you can recognise on the image

Two Images

This allows the pixel coordinates from one image to be translated into the pixel coordinates in a second image. This is useful if you need to overlay two images such as an x-ray fluorescence image over a reflected light images

What you need:

- two images you wish to overlay
- 3 features you can recognise on both images



2.3.2 Workflows

This set of pages walks you through how to use **autopew** for each use case.

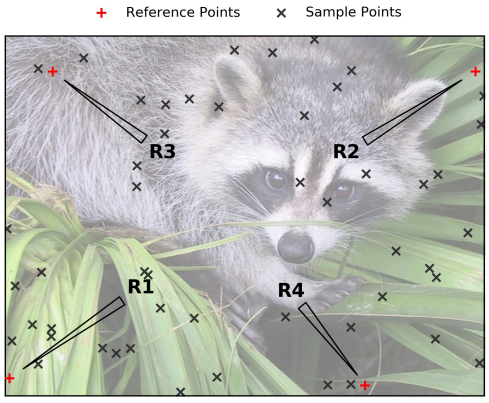
Image to Stage

This is a simple workflow example involving converting points registered on an image to coordinates for a specific stage.

INPUT: image of sample

OUTPUT: scancsv file in laser coordinate system with corresponding spotnames + realigned image

e.g we have been using this method to extract phases from reflected light imagery or X-ray fluorescence mapping and assign numbers to grains to give spatial context to the multiple different micro-geochemical analysis.



Step 1: Acquire an Image and Register Points

- Aquire an image of your sample
- add points to your image

Once an image is acquired, points can be added (either via ImageJ or Fiji*[0] , or through **autopew** extensions). If you use ImageJ, export your points as a .csv file and follow the Stage to stage workflow which outlines transforming a list of X,Y coordinates into a new translation. The following workflow is designed using the **autopew** extensions.

For selection points directly in **autopew** here is an example:

```
import numpy as np
import pandas as pd
from pathlib import Path
from autopew.workflow import pick_points
from autopew.workflow.laser import points_to_scancsv
from autopew.transform.affine import affine_transform, affine_from_AB

# have an image you wish to use?
imagepath = Path("./../../source/_static/") / "img.jpg"

# pick coordinates from an image you have handy?
pixel_sample_coords = pick_points(imagepath)
```

Step 3: Calibrate the Transformation between the Image and Stage

- Pick a 3 or more calibration points

Note that the calibration of this transform involves a least-squares process to find the optimal transformation, such that adding more calibration points can help avoid minor inaccuracies in adding points.

Step 4: Transform Image Point Coordinates to Stage Coordinates

- give the same reference points in the stage Coordinates (laser reference coordinates)
- Use **autopew** to transform all pixel coordinates to stage coordinates. See the example code below:

```
# %% LASER REFERENCE POINTS ------------------------------------------------------
↪--
laser_reference_coords = np.array([
      [74978,85419],
      [90828,82571],
      [80259,75389],
      [81465,74373]])


# %% CALCULATE TRANSFORM ---------------------------------------------------------
↪--
transform = affine_transform(
  affine_from_AB(pixel_reference_coords, laser_reference_coords)
)
# %% TRANSFORM SAMPLE POINTS -----------------------------------------------------
↪--
```

(continues on next page)

[0] Schneider, C. A.; Rasband, W. S. & Eliceiri, K. W. (2012), "NIH Image to ImageJ: 25 years of image analysis", Nature methods 9(7): 671-675, PMID 22930834

(continued from previous page)

```
# these are the magic points we want
laser_sample_coords = transform(pixel_sample_coords)

# %% Visualise the Transform
from autopew.util.plot import plot_transform

fig = plot_transform(
  pixel_sample_coords,
  tfm=transform,
  ref=pixel_reference_coords,
  invert0=[False, True],
  invert1=[False, True],)
```

Step 5: Export Points to for Stage Coordinates

- Export the transformed point stage coordinates to a file you can import into the software controlling the stage.

```
# %% EXPORT to .Scancsv file ------------------------------------------------------
↪--
# lets save them so we can directly import them
points_to_scancsv(
  laser_sample_coords, filename="output_filename", spotnames=spotnames
)
```

See also:

output types

Optional Next Steps

- Export an aligned image.

Imported images can be realigned to the stage coordinate system for easier recognition of sample features and more accurate visual determination of new point location.

References

Stage to Image

This is a simple workflow example involving converting points from one stage coordinate system to display on a large image. This helps to visualise where the analysis were collected and gives context to the anaysis.

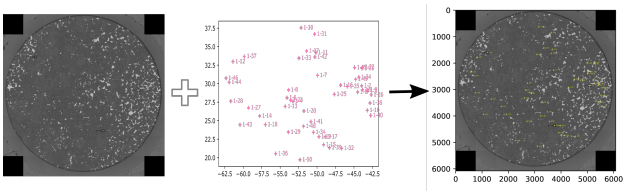
INPUT:

- .csv - a list of X,Y coordinates with spotnames + more than 3 reference points
- large image with locations of more than 3 reference points

OUTPUT:

- image with points labelled

e.g. We have been using the workflow to visualise the microanalysis points from SEM on large reflected light images



Step 1: Acquire image

- collect an image of the sample
- Remember to highlight 3 regions or more for registration points

Step 2: Calibrate and Transform the points between the the image and the stage

- Import your CSV file with analysed points
- specify your >3 reference coordinates using the **autopew** interacitve interface
- specify the stage Coordinates of these reference points
- Use **autopew** to transform all stage coordinates. See the example code below:

Note that the calibration of this transform involves a least-squares process to find the optimal transformation, such that adding more calibration points can help avoid minor inaccuracies in adding points.

```
import numpy as np
import pandas as pd
from pathlib import Path
from autopew.workflow import pick_points
from autopew.workflow.laser import points_to_scancsv
from autopew.transform.affine import affine_transform, affine_from_AB

# let the code know what folder we are working in:
start_here = Path("./../../source/_static/")

# %% INPUT STAGE ALL POINTS ------------------------------------------------------------
↪-----
#import the coordinate from the source stage
df = pd.read_csv('Analysis_Points.csv')
#drop any blank rows
df = df.dropna(how='all', axis='index')
#add the names of the points
spotnames=df["ID"]
#tell the code what the X,Y columns are named
stage_sample_coords = np.array([df["X"],df["Y"]]).T

# %% INPUT REFERENCE POINTS ------------------------------------------------------------
↪--
# Pick reference points from the displayed image
```

(continues on next page)

(continued from previous page)

```
imagepath = start_here / "img.jpg"
pixel_reference_coords = pick_points(imagepath)

# %% STAGE REFERENCE POINTS ------------------------------------------------------------
↪--
#Enter the coordinates for the reference points you chose
stage_reference_coords = np.array([[-1300,-6120],  #R1
                                   [-8460,-1410], #R2
                                   [-1960,-1420], #R3
                                   [-6770,-6240]]) #R4

# %% CALCULATE TRANSFORM ---------------------------------------------------------------
↪--
transform = affine_transform(
  affine_from_AB(stage_reference_coords, pixel_reference_coords)
)
# %% TRANSFORM SAMPLE POINTS -----------------------------------------------------------
↪--
# these are the magic points we want
pixel_sample_coords = transform(stage_sample_coords)
```

Step 4: Overlay the image and the points

- Export an image containing labelled point overlay over image

```
# %% GIVE NAMES TO THE NEW POINTS ------------------------------------------------------
↪-------
new_coords=pd.DataFrame(data=pixel_sample_coords[0:,0:], columns=['x','y'])
new_coords["ID"]=spotnames

# %% FIND THE PIXEL SIZE OF THE IMAGE--------------------------------------------------
↪--------
from PIL import Image

img = Image.open(imagepath)
# get the image's width and height in pixels
width, height = img.size

# %% PLOT THE POINTS ON THE IMAGE------------------------------------------------------
↪----
import matplotlib.pyplot as plt

X=new_coords['x']
Y=new_coords['y']
label=new_coords['ID']

fig, ax = plt.subplots()
ax.scatter(X, Y,color='yellow', marker="+",zorder=1,s=6,linewidth=.3)

for i, df in enumerate(label):
    ax.annotate(df, (X[i], Y[i]),
                xytext=(2, 0), textcoords='offset points',
                horizontalalignment='left', verticalalignment='center',
                size=2, color='yellow',
                zorder=1)
```

(continues on next page)

(continued from previous page)

```
ax.set(xlim=(0, width), ylim=(0, height))

plt.imshow(img, zorder=0)
ax.invert_yaxis()#image invert so it is the same up direction as import.

plt.show()
#fig.savefig('temp.png', transparent=True, dpi=800) #Optional Image Export
```

See also:

output types

Stage to Stage

This is a simple workflow example involving converting points from one stage coordinate system to another. This workflow also works for importing .csv files of pixel x,y coordinates from software such as imageJ.

INPUT: .csv - a list of X,Y coordinates with spotnames + more than 3 reference points

OUTPUT: scancsv file in laser coordinate system with corresponding spotnames

e.g. We have been using the workflow to ensure measurement of the same grain on both scanning electron microscope and the laser ablation system.

Step 1: Acquire coordinates

- save the coordinates of 3 or more registration points
- collect and save coordinates of phases of interest

Step 2: Export Point Coordinates to CSV

- Export the stage-coordinates of your planned points in X-Y(-Z) format to a CSV.

Optionally, specify names for each of the points, which could be used as an index later.

Step 3: Calibrate and Transform the points between the two stages

- Import your origin CSV file
- specify your >3 reference coordinates
- give the same points in the new stage Coordinates
- Use **autopew** to transform all stage coordinates. See the example code below:

Note that the calibration of this transform involves a least-squares process to find the optimal transformation, such that adding more calibration points can help avoid minor inaccuracies in adding points.

```
import numpy as np
import pandas as pd
from pathlib import Path
from autopew.workflow import pick_points
from autopew.workflow.laser import points_to_scancsv
```

(continues on next page)

(continued from previous page)

```
from autopew.transform.affine import affine_transform, affine_from_AB

# let the code know what folder we are working in:
start_here = Path(r"C:\FILEPATH")

# First we Need Some Coordinates.
# %% INPUT STAGE ALL POINTS ------------------------------------------------------------
↪-----
#import the coordinate from the source stage
df = pd.read_csv(start_here/"LaserPoints.csv")
#drop any blank rows
df = df.dropna(how='all', axis='index')
#add the names of the points
spotnames=df["PointName"]
#tell the code what the X,Y columns are named
pixel_sample_coords = np.array([df["X"],df["Y"]]).T

# %% INPUT STAGE REFERENCE POINTS ------------------------------------------------------
↪--------
pixel_reference_coords = np.array([
      [-26369.84,-35504.5],  #R1
      [-10899.56,-34236.77], #R2
      [-18028.68,-25400.77], #R3
      [-18679.94,-25251.63]])#R4

# %% OUTPUT STAGE REFERENCE POINTS -----------------------------------------------------
↪---------
laser_reference_coords = np.array([
      [74978,85419],   #R1
      [90828,82571],   #R2
      [80259,75389],   #R3
      [81465,74373]])  #R4


# %% CALCULATE TRANSFORM ---------------------------------------------------------------
↪--
transform = affine_transform(
  affine_from_AB(pixel_reference_coords, laser_reference_coords)
)
# %% TRANSFORM SAMPLE POINTS -----------------------------------------------------------
↪--
# these are the magic points we want
laser_sample_coords = transform(pixel_sample_coords)

# %% Visualise the Transform
from autopew.util.plot import plot_transform

fig = plot_transform(
  pixel_sample_coords,
  tfm=transform,
  ref=pixel_reference_coords,
  invert0=[False, True],
  invert1=[False, True],)
```

Step 4: Export Points to for new Stage Coordinates

- Export the transformed point stage coordinates to a file you can import into the software controlling the new stage.

```
# %% EXPORT to .Scancsv file ------------------------------------------------------
↪--
# lets save them so we can directly import them
points_to_scancsv(
  laser_sample_coords, filename="output_filename", spotnames=spotnames
)
```

See also:

output types

2.3.3 Outputs

Here we outline the file formats currently available for export in **autopew**

Output Types

The output types can be edited to suit the equipment you wish to use. The current output of coordinates are designed for the following analytical equipment.

Laser Ablation system

The .scancsv current output is compatible with a Photonmachines (Teledyne) laser running on Chromium. This allows for rapid input of X,Y coordinates into the laser system with spot labels either read from the source file, or numbered sequentially.

Currently all other settings; including focus (z), spot size and laser conditions need to be changed manually after import into the laser.

TESCAN SEM

work in progress The TESCAN SEM allows the import of X,Y coordinates via .csv files.

2.4 API

2.4.1 autopew.transform

Submodule for calculating affine transforms between planar coordinate systems.

autopew.transform.affine

Submodule for calculating affine transforms between planar coordinate systems.

autopew.transform.affine.**affine_from_AB**(*X*, *Y*)
Create an affine transforamtion matrix based on two sets of coordinates.

Note:
- This is an augmented matrix, and includes the translation component

---

autopew.transform.affine.**affine_transform**(*A*)
Create an affine transform function based on affine matrix A.

autopew.transform.affine.**compose_affine2d**(*T=array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]]), Z=array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]]), R=array([[1., 0., 0.], [-0., 1., 0.], [0., 0., 1.]])*)
Compose an affine transformation matrix based on translation, zoom and rotation components.

Parameters **T, Z, R** (numpy.ndarray) – Component affine transfrom matricies for translation, zoom/scaling and rotation.
Returns **A**
Return type numpy.ndarray

autopew.transform.affine.**corners**(*size*)

autopew.transform.affine.**decompose_affine2d**(*A*)
Decompose an affine transform into components using a polar transform.
Returns **T, Z, R**
Return type numpy.ndarray

Note: This decomposes the transform into the sequence rotation - zoom - translation.

To recompose this transform,

---

autopew.transform.affine.**rotate**(*theta=0*, *degrees=True*)
Generate a 2D affine rotation matrix.

Uses clockwise rotations.

autopew.transform.affine.**shear**(*x=0*, *y=0*)
Generate a 2D affine shear matrix.

autopew.transform.affine.**translate**(*x=0*, *y=0*)
Generate a 2D affine translation matrix.

autopew.transform.affine.**zoom**(*x=1*, *y=1*)
Generate a 2D affine zoom matrix.

2.4.2 autopew.image

autopew.image.registration

2.4.3 autopew.graph

autopew.graph.network

class autopew.graph.network.**Net**
Network of transformations between objects.
This object stores the individual node objects including their properties and registration points, in addition to the edges which involve specific coordinate transforms.

add_edge(*A*, *B*, *transform=None*, ***kwargs*)
Add an edge between components A and B. Optionally specify the specific transform.

Parameters
- **A, B** (str) – Names of components to link.
- **transform** (function) – Function to transform coordinates from A space to B space.
- **inverse_transform** (function) – Function to transform coordinates from B space to A space.

draw(*ec='k'*, *nc='seagreen'*, *ax=None*, *figsize=(10, 10)*, *method=<function draw_shell>*)

edges

get_transform(*A*, *B*)
Get the function to transform coordinates between nodes A and B.

link(*A*, *B*, *transform=<function autolink>*, *inverse_transform=<function autolink>*, ***kwargs*)
Link nodes A and B with transforms along edges.

nodes

update(*name*, *obj*, ***kwargs*)

autopew.graph.network.**autolink**(*x*)
Default link function.

2.4.4 autopew.io

autopew.io.laser.readlase

autopew.io.laser.writelase

2.4.5 autopew.gui

autopew.gui

2.4.6 autopew.util

autopew.util.plot

autopew.util.plot.**bin_centres_to_edges**(*centres*)
Translates point estimates at the centres of bins to equivalent edges, for the case of evenly spaced bins.

autopew.util.plot.**bin_edges_to_centres**(*edges*)
Translates edges of histogram bins to bin centres.

autopew.util.plot.**data_to_pixels_transform**(*ax*)

autopew.util.plot.**plot_2dhull**(*data*, *ax=None*, *s=0*, ***plotkwargs*)
Plots a 2D convex hull around an array of xy data points.

autopew.util.plot.**plot_transform**(*src*, *dest=None*, *tfm=None*, *ref=None*, *ax=None*, *sharex=False*, *sharey=False*, *invert0=[False, False]*, *invert1=[False, False]*, *figsize=(10, 5)*, *titles=['Source Coordinates', 'Destination Coordinates']*, *hull=True*)
Visualise an affine transfrom.

autopew.util.meta

autopew.util.meta.**autopew_datafolder**(*subfolder=None*)
Returns the path of the autopew data folder.

Parameters **subfolder** (str) – Subfolder within the autopew data folder.

Returns

Return type pathlib.Path

autopew.util.meta.**chain**(*lst*)
Chain a series of fuctions together.

2.5 Development

autopew is currently hosted on GitHub at github.com/morganjwilliams/autopew; collaborator access can be granted to interested parties. If you're new to Git or GitHub, there are some useful guides on the GitHub Website.

2.5.1 Development Installation

To access and use the development version, you can either clone the repository or install via pip directly from GitHub:

```
pip install git+git://github.com/morganjwilliams/autopew.git@develop#egg=autopew
```

2.5.2 Branches and GitFlow

There are two main git-branches for autopew:

- master is the latest stable release.
- develop is the development branch.

The Git workflow is based on GitFlow, where releases are branched from develop prior to being integrated into master. Pull requests should be made against the develop branch.

2.5.3 Documentation

Documentation is currently live on *ReadtheDocs.org <https://autopew.readthedocs.io>*‘__ , but can also be built and viewed locally using instructions below. The documentation is built using sphinx, and most pages are written in reStructuredText. A quick reference can be found here.

Documentation for `autopew` is in the `docs` directory. From this directory, documentation can be built as follows:

To build documentation on windows:

```
# to build and view the html version:
make html && cd ./build/html/ && index.html && cd ../..
# or, to build and view the latex-pdf version:
make latex && cd ./build/latex/ && make.bat && autopew.pdf && cd ../..
```

Alternatively, there is a default build batch file `makeviewhtml.bat` also located in the docs directory, which executes the commands above and will automatically build the docs and open the landing page:

```
# to build and view the html version:
makeviewhtml.bat
```

2.5.4 Tests

If you clone the source repository, unit tests can be run using pytest from the root directory after installation:

```
python setup.py test
```

2.5.5 Continuous Integration

There are also some active continuous integration tools for `autopew`, including automated unit-testing on Travis-CI and test coverage analysis on Codecov. The details for each of these are listed below.

Travis-CI

The Travis-CI page for `autopew` can be found at travis-ci.org/morganjwilliams/autopew.

Code Coverage

The coveralls page for `autopew` can be found at coveralls.io/github/morganjwilliams/autopew.

2.6 Changelog

All notable changes to this project will be documented here.

2.6.1 Development

Note: Changes noted in this subsection are to be released in the next version. If you’re keen to check something out before its released, you can use a development install.

2.6.2 0.0.1

2.7 Future

This page details some of the under-development and planned features for **autopew**. Note that while no schedules are attached, features under development are likely to be completed with weeks to months, while those 'On The Horizon' may be significantly further away (or in some cases may not make it to release).

Initial development focuses on work relevant to LA-ICP-MS workflows, and the project may later generalise some of these workflows.

2.7.1 Current Release

- Able to register points on one image
- Able to calibrate these points to the laser stage coordinates
- Able to output the image with points on it
- Able to output a .scancsv file to import into the laser software
- Able to import point names

2.7.2 Under Development

- Be able to set spotsizes
- Be able to read and set z position
- Be able to recognise when point is positioned outside stage limits
- Be able to transform maps with points on them so correspondence is easier to see
- Should be able to work from multiple maps –> need multiple registered images
- Overlay images (see post here)
- Serialising coordinate transform systems for later use

2.7.3 On The Horizon

- Activities to optimise instrument usage time:
  - Sample-standard bracketing using a specific 'reference mount location' (needs to be updated later)
  - Reordering points based on their positions.
  - Standard area registration, gridded 'free positions'- auto sample standard bracketing
- Report templates
- Dealing with larger images (>20 MB - jpg will warn of compression bomb)

2.8 Code of Conduct

2.8.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

2.8.2 Our Standards

Examples of behaviour that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behaviour by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

2.8.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behaviour and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behaviour.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviours that they deem inappropriate, threatening, offensive, or harmful.

2.8.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

2.8.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behaviour may be reported by contacting the project admins. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

2.8.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant Version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html. The Contributor Covenant is released under the Creative Commons Attribution 4.0 License.

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq.

2.9 Contributing

`autopew` welcomes community contributions of all forms. Requests for features and bug reports are particularly valuable contributions, in addition to code and expanding the documentation.

All individuals contributing to the project are expected to follow the Code of Conduct, which outlines community expectations and responsibilities.

Also, be sure to add your name or GitHub username to the contributors list.

Note: This project is currently in *alpha*, and as such there's much work to be done.

2.9.1 Feature Requests

If you're new to Python, and want to implement something as part of `autopew`, you can submit a Feature Request. Perhaps also check the Issues Board first to see if someone else has suggested something similar (or if something is in development), and comment there.

2.9.2 Bug Reports

If you've tried to do something with `autopew`, but it didn't work, and googling error messages didn't help (or, if the error messages are full of `autopew.XX.xx`), you can submit a Bug Report . Perhaps also check the Issues Board first to see if someone else is having the same issue, and comment there.

2.9.3 Contributing to Documentation

The documentation and examples for `autopew` are gradually being developed, and any contributions or corrections would be greatly appreciated. Currently the examples are patchy, and a 'getting started' guide would be a helpful addition. If you'd like to edit an existing page, the easiest way to get started is via the 'Edit on GitHub' links:

These pages serve multiple purposes:

- A human-readable reference of the source code (compiled from docstrings).
- A set of simple examples to demonstrate use and utility.
- A place for developing extended examples

2.9.4 Contributing Code

Code contributions are always welcome, whether it be small modifications or entire features. As the project gains momentum, check the Issues Board for outstanding issues, features under development. If you'd like to contribute, but you're not so experienced with Python, look for `good first issue` tags or email the maintainer for suggestions.

To contribute code, the place to start will be forking the source for `autopew` from GitHub. Once forked, clone a local copy and from the repository directory you can install a development (editable) copy via `python setup.py develop`. To incorporate suggested changes back to into the project, push your changes to your remote fork, and then submit a pull request onto autopew/develop .

Note:

- See Installation for directions for installing extra dependencies for development, and Development for information on development environments and tests.
- `autopew` development roughly follows a gitflow workflow. `autopew/master` is only used for releases, and large separable features should be build on `feature` branches off `develop`.
- Contributions introducing new functions, classes or entire features should also include appropriate tests where possible (see *Writing Tests*, below).
- `autopew` uses Black for code formatting, and submissions which have passed through `Black` are appreciated, although not critical.

2.9.5 Writing Tests

There is currently a minimal unit test suite for `autopew`, which guards against breaking changes and assures baseline functionality. `autopew` uses continuous integration via Travis, where the full suite of tests are run for each commit and pull request, and test coverage output to Coveralls.

Adding or expanding tests is a helpful way to ensure `autopew` does what is meant to, and does it reproducibly. The unit test suite one critical component of the package, and necessary to enable sufficient trust to use `autopew` for scientific purposes.

2.10 Contributors

This list includes people who have contributed to the project in the form of code, comments, testing, bug reports, feature requests.

- Louise Schoneveld
- Morgan Williams

Note: This documentation is a work in progress and is updated regularly. Contact the maintainers with any specific questions/requests.

CHAPTER 3

References

Python Module Index

a

Index