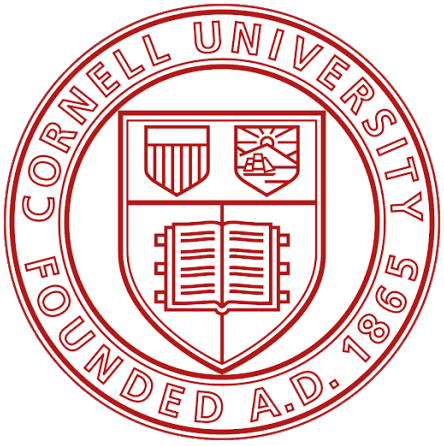


Cornell University
Cornell High Energy
Synchrotron Source
Ithaca, NY

Fort Lewis College
Department of
Physics and Engineering
Durango, CO

Python-based Image Processing for In-situ X-ray Imaging

Author:
John Lee

Advisors:
Arthur Woll
Louisa Smieska

August 3, 2020
Summer 2020

Abstract

The purpose of this project was to create online web pages to serve as walkthroughs for tutorials on how to conduct a flatfield correction and perform particle image velocimetry from full-field x-ray data, as well as an introduction page for the FMB beamline and and overview on full-field image collection. This was done by creating wiki pages hosted by CLASSE at Cornell and using Python programming language within Jupyter Notebooks to alter the data and perform the image processing techniques.

1 Background

Due to COVID-19, having the ability to work remotely has become of utmost importance. The Functional Materials Beamline (FMB) at MSN-C was built in part to provide x-ray-based techniques for improved materials processing. In order to make these tools more broadly accessible, this project's aim was to create documentation for the operations and techniques used at the beamline.

Creating documentation for the types of image collection procedures done at FMB can aid in saving time to train new employees and faculty about the basics of x-ray image collection. It allows new users to learn the terminology and functions of the beamline without requiring resources from existing personnel. Storing this documentation online makes it easy to access, as well as update.

In addition to informational web pages, there is also value in having standalone tutorials for users to access so they may practice the science on their own without having a formal training session. To aid in understanding the tutorials, walkthroughs are desirable to help the user have a better grasp on the material and give a feel for what is being done.

2 Objectives

The goal of the this project was to generate web pages as documentation for the functions of FMB and describe how data from the beamline is collected and processed. The main focus of the web pages was on making walkthroughs for tutorials on flatfield correction and particle image velocimetry (PIV), both of which are image processing procedures for data collected by full-field imaging. Other pages were also created, including an introduction page for FMB and a guide on how to install the required software to conduct the tutorials for the image processing techniques described above. Another objective of this project was to alter the PIV tutorial in order to make it more user-friendly by lessening the workload required by the user, making it more of a standalone process.

3 Outcomes

The sections below give titles and descriptions of the web pages that were generated, and brief overviews of the information stored on each page.

3.1 Introduction page

This page serves as an introduction for FMB and it's specific functions regarding the types of materials and specifications of the x-rays that are used. It includes a link to a virtual tour of the beamline area and a detailed description of the equipment, as well as the image collection methods that are used in the hutch.

3.2 How to Install Python

This page includes instructions on how to install the Python programming software and integrated development environments (IDEs). The page also includes a recommendation section that refers users to download Anaconda (a Python platform manager) so they have access to the Jupyter Notebooks application, which is required to run the image processing tutorials.

Python is open-sourced free software programming language that is very powerful and can be used in a variety of ways, from webpage design to data analysis. Anaconda is a program that comes with many

pre-downloaded applications that are useful in processing and reporting scientific information, such as the Spyder IDE and JupyterLabs applications.

3.3 Full-field Imaging

This page provides information about the full-field image collection technique and how it is used in everyday applications. It also includes information about how full-field imaging is used to collect data from x-rays at the beamline, such as examples of typical setups for the equipment and examples of images that are produced. This page links to another page that talks about how to process the x-ray data, which is described in the section below. Figure 1 was created using the full-field imagining using visible light, and Figure 3 (left) is an example of full-field imaging using x-rays. Figure 2 shows an example of the equipment setup of full-field image collection for x-rays, where the "camera" (detector) is fully in the path of the x-rays for the images it is capturing.

Figure 1: This image was taken using the full-field imaging technique, where all the pixels were collected at the same time. [Source: unsplash.com]

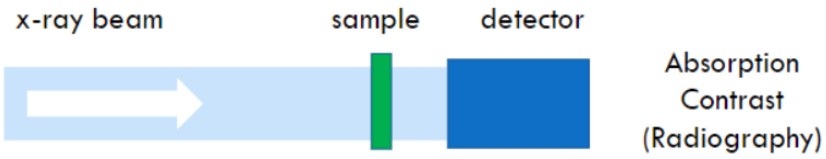


Figure 2: This is an example of a common setup used in full-field imaging using x-ray beams.

Full-Field Imaging is any imaging technique in which all pixels of the image are collected at the same time, in parallel. This is in contrast to the more detailed and time-consuming Scan-Probe imaging technique, where information from multiple positions is collected sequentially and then compiled to produce a complete image. Full-field imaging is the most common image collection technique; when a picture is taken with a digital camera, it uses this method to capture the image.

3.4 Image Processing Techniques

This page was created to host the tutorials on flatfield correction and particle image velocimetry, and was the main focus of this project. The tutorials are available to be downloaded on the page and include complete data sets, making it a one-stop shop for any person that wishes to actually conduct the image processing techniques themselves.

3.4.1 Flatfield Correction

This section of the image collection technique webpage includes a description of what the flatfield correction is, as well as show an example of how an image looks before and after the correction is performed. It also gives two versions of an overview of the steps that are done within the tutorial, as well as the files for the tutorial itself.

Flat-field image correction refers to the elimination of image contrast due to variations in the incident beam intensity in different regions of an image. An example of this completed process is shown in Figure 3, with each pixel measuring approximately 1.2 microns. Notice how the detail within the corrected picture can be seen much easier, making it feasible for use in analysis.

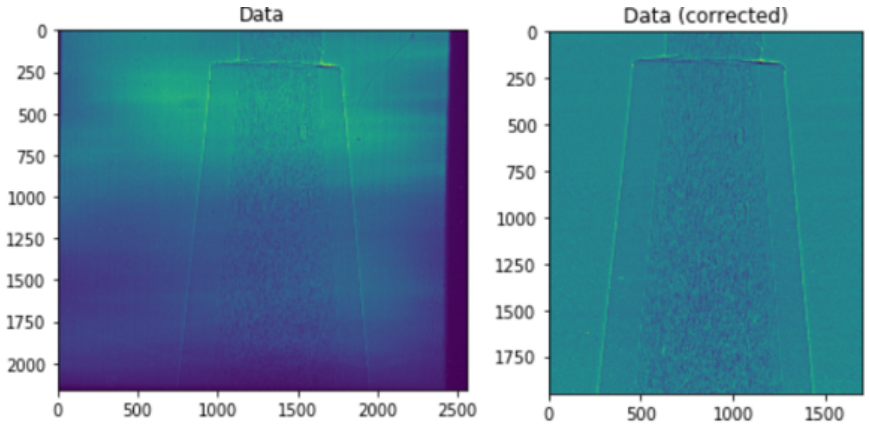


Figure 3: (Left) Raw image data from the beamline of a 3D printer nozzle during printing of epoxy filled with chopped silicon carbide fiber. (Right) The same image after performing a flatfield correction and cropping the image.

3.4.2 Particle Image Velocimetry

Much like the flatfield walkthrough, this section of the web page includes a description of the procedure as well as potential uses for its application. It includes two versions of walkthroughs for the steps in the tutorial and the files for the tutorial itself.

Particle Image Velocimetry is a method of visualizing flows from radiographs by creating a vector field. It can be used to produce instantaneous velocity measurements within a flow, given there are particles in the fluid that can be used to measure displacement. By knowing the displacement of the particles between frames in combination with the time between each frame, velocity vectors can be calculated for each particle (velocity = displacement / time). This can be helpful in determining potential causes of clogs in a pipe, for example: while 3D printing. Figure 4 is an example of PIV analysis, and the units are displacement of pixels per frame.

In addition, the tutorial for PIV was streamlined by integrating code that allowed users to automatically create the "mask" of an image to be used in analysis by generating an interactive interface on the images being processed, as seen in Figure 5. The purpose of the mask is to denote the area to be analyzed for the flowing particles, giving the software a "window" to look in to measure displacement of particles. The code used to generate the masks was obtained from Github [1]. This saved the user from having to generate their

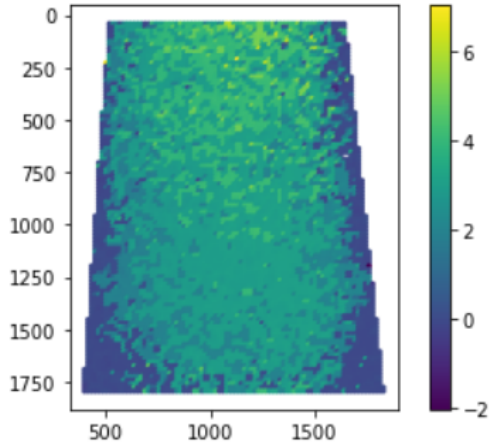


Figure 4: Output from the PIV tutorial. The points represent velocity vectors.

own mask image using another software such as Microsoft Paint, saving the user much time and decreasing the front-end work required to run the tutorial.

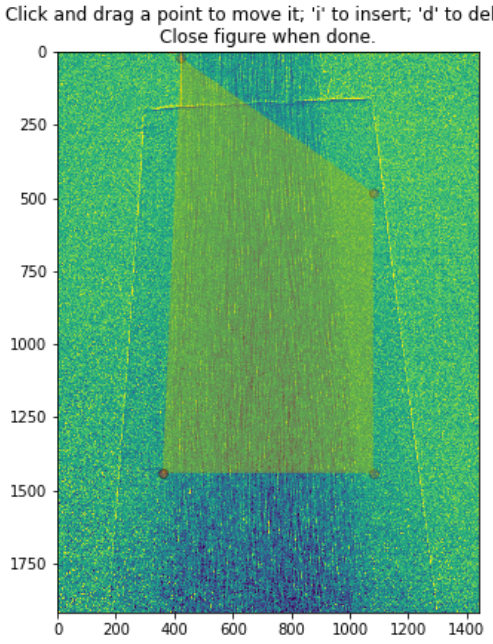


Figure 5: The interactive mask generator software integrated into the PIV tutorial. The user can drag the red points to outline the area of the image to be analyzed.

4 Future Directions

Currently the flat-field correction tutorial includes example data, making it such that anyone can download the tutorial file and test it on their home computer. As of the time of this report, the PIV tutorial does not have the flatfield-corrected data that was used to create Figure 4, however it does have an example data set that can still produce an image that can give the user a feel for the process. An immediate improvement

would be to correct the data originally used and alter the "window size" parameters so that the image in Figure 4 can be produced by users from their home computers.

The web pages are a continuously evolving document, so it is anticipated that their content will be updated as more techniques become added and validated at FMB.

Acknowledgements

John Lee would like to acknowledge Dr. Arthur Woll and Dr. Louisa Smieska for their assistance in learning Python and the background information on the functions of the beamline, as well as Brendan Croom for creating the tutorials and documenting the steps it takes for the image processing procedures. John would also like to acknowledge support from Fort Lewis College and Cornell University from Dr. Laurie Williams and Dr. Matthew Miller for presenting this opportunity.

This work was supported by the Center for High Energy X-ray Sciences (CHEXS) which is supported by the National Science Foundation under award DMR-1829070 and the Materials Solutions Network at CHESS (MSN-C) which is supported by the Air Force Research Laboratory under award FA8650-19-2-5220.

References

1. Barnes, R. *Tool to create polygon mask in Matplotlib* https://gist.github.com/r-barnes/. 2017.

Appendix A

Flat-field correction tutorial

Developer: Brendan Croom (brendan.croom.ctr@us.af.mil)

Date: May 26, 2020

In this tutorial, we will go over the following:

- Loading and exploring HDF data using h5py
- Plotting radiographs using Matplotlib / Pyplot
- Flat-field correction of radiographs
- Conveting a series of images to a video (MP4 format)

To begin, we need to load a few libraries and define where the data is saved.

In [1]:
```
# Load the appropriate libraries:
import os
from matplotlib import pyplot as plt
import numpy as np
import h5py
import h5py_radio_processing as radio

# Where is the data saved?
data_folder = 'xray_data'
dark_file = 'cf10to1_darks_ANDOR2_001_0000.hdf'  # contains a sequence of imag
es
white_file = 'cf10to1_whites_ANDOR2_001_0000.hdf'  # also contains a sequence
 of images
data_file = 'cf10to1_mosaic_p16.5_1_ANDOR2_001_0000.hdf'  # contains only a si
ngle image
```

Loading example data

First, we read HDF data and explore the file contents. The HDF is a container, and is organized like a file directory. We want to find the path that contains the image data.

In [2]:
```
# Load an example image, then explore the contents
hdf_data = h5py.File(os.path.join(data_folder, data_file), 'r')

# Explore the contents of the HDF.
radio.map_hdf(hdf_data)
```

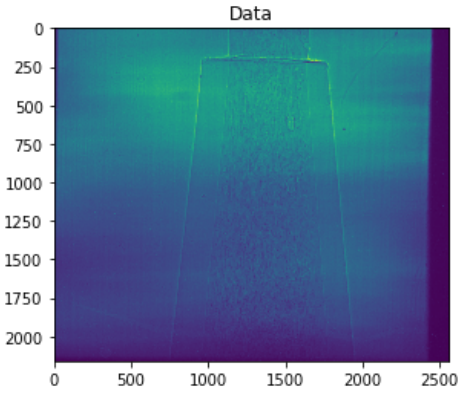
```
| 'entry': type <class 'h5py._hl.group.Group'>
| | 'instrument': type <class 'h5py._hl.group.Group'>
| | | 'NDAttributes': type <class 'h5py._hl.group.Group'>
| | | 'detector': type <class 'h5py._hl.group.Group'>
| | | | 'NDAttributes': type <class 'h5py._hl.group.Group'>
| | | | 'data': type <class 'h5py._hl.dataset.Dataset'>
| | | 'performance': type <class 'h5py._hl.group.Group'>
| | | | 'timestamp': type <class 'h5py._hl.dataset.Dataset'>
```

In the previous step, we observed that the data is located at the following location within each HDF container:

In [3]:
```
DATA_LOCATION = 'entry/instrument/detector/data'
```

Plot radiographs using Pyplot

In [4]:
```
data_image = hdf_data[DATA_LOCATION]

f, ax = plt.subplots()
ax.imshow(data_image)
ax.set_title('Data')
plt.show()
```

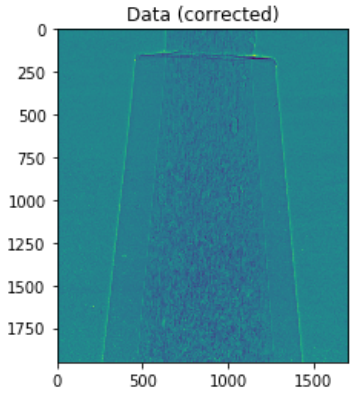


You'll notice two key features about the image, which we want to correct:

- First, the intensity of the image is uneven. This is due to variation in the beam intensity across the field of view, and can be fixed using a flat-field correction.
- Second, the specimen does not fill the field of view. We can crop the image to remove un-used portions of the image.

When we load the white and dark images, note that they contain a series of images. They are averaged to reduce image noise.

In [5]:
```
# Slice the data to remove un-used areas of the image
DATA_CROP = (slice(50, 2000), slice(500, 2200))

# Load the dark and white images. Note that the
data_dark_full = h5py.File(os.path.join(data_folder, dark_file), 'r')[DATA_LOC
ATION]
data_white_full = h5py.File(os.path.join(data_folder, white_file), 'r')[DATA_L
OCATION]

data_dark = np.average(data_dark_full, axis=0)
data_white = np.average(data_white_full, axis=0)

# Apply a flat-field correction
image_flat = radio.flatfield_correction(data_image, data_white, data_dark)

f, ax = plt.subplots()
ax.imshow(image_flat[DATA_CROP])
ax.set_title('Data (corrected)')
plt.show()
```



Rendering an MP4 video

For the next step, we'll process a series of radiographs and export as a video. Each frame needs to be flat-field corrected, saved as a TIFF, then added to a video. Note that t_exposure for the data was 0.03 seconds / frame, versus 0.5 seconds for the white and dark images.

For this, we'll use a new data set, and we'll have to import a few new libraries.

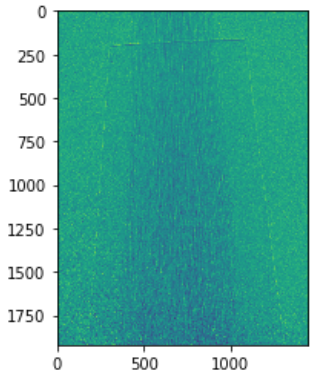
In [6]:
```
import imageio as io

# Where is the data saved?
data_folder = 'xray_data'
dark_file_sic = 'sic10to2_darks_z7.5n2_ANDOR2_001.hdf'
white_file_sic = 'sic10to2_whites_z7.5n2_ANDOR2_002.hdf'
data_file_sic = 'sic10to2_movie_z0n2_p5.5_1_ANDOR2_001.hdf'

# Where to save the data?
outfolder = 'sic_images'

# Apply a crop to the data:
DATA_CROP_SIC = (slice(80, 2000), slice(460, 1900))

exposure_ratio = 0.5 / 0.03
```

As before, we'll load and average the "white" and "dark" images:

In [7]:
```
data_dark_sic_full = h5py.File(os.path.join(data_folder, dark_file_sic), 'r')[
DATA_LOCATION]
data_white_sic_full = h5py.File(os.path.join(data_folder, white_file_sic), 'r'
)[DATA_LOCATION]
data_dark_sic = np.average(data_dark_sic_full, axis=0)[DATA_CROP_SIC]
data_white_sic = np.average(data_white_sic_full, axis=0)[DATA_CROP_SIC]
```

First, we'll load a test image from the data set to verify that the flat-field correction works.

In [8]:
```
data_test = h5py.File(os.path.join(data_folder, data_file_sic), 'r')[DATA_LOCATION][0]

data_flat = radio.flatfield_correction(data_test[DATA_CROP_SIC], data_white_sic, data_dark_sic, exposure_ratio)
plt.imshow(data_flat)
plt.show()
```



Next, we'll process each image located in the data file, and save as a TIFF.

In [9]:
```
exposure_ratio = 0.5 / 0.03
with h5py.File(os.path.join(data_folder, data_file_sic), 'r') as data:
    for i, img in enumerate(data[DATA_LOCATION]):
        # Apply flat field correction
        data_flat = radio.flatfield_correction(
            img[DATA_CROP_SIC], data_white_sic, data_dark_sic,
            exposure_ratio)

        # Rescale to 8bit image
        data_flat = radio.scale_to_8bit(data_flat, 0.3, 1.2)

        # Save as a TIFF
        try:
            io.imsave(os.path.join(outfolder, 'image_{:04d}.tiff'.format(i)),
                      data_flat
                     )
        except FileNotFoundError:
            os.makedirs(outfolder)
            io.imsave(os.path.join(outfolder, 'image_{:04d}.tiff'.format(i)),
                      data_flat
                     )
```

Finally, we'll read the TIFF images and save in MP4 format.

In [10]:
```
image_list = [f for f in os.listdir(outfolder) if f.endswith('.tiff')]
with io.get_writer('sic_fiber_video.mp4', fps=10) as writer:
    for file in image_list:
        img = io.imread(os.path.join(outfolder, file))
        # Use only half resolution so the video is of reasonable size
        writer.append_data(img[::2, ::2])
```

Appendix B

PIV_introduction-Copy1

August 5, 2020

1 Introduction to PIV code

Brendan Croom brendan.croom.ctr@afresearchlab.com / b.p.croom@gmail.com

This notebook is intended to introduce the custom Particle Image Velocimetry (PIV) code used to quantify fluid flow from a series of X-ray radiographs. It was originally developed to study the flow of a fiber-containing resin during extrusion through a nozzle during 3D printing. It assumes some familiarity with PIV, which uses correlation algorithms to track the displacement between images. The Wikipedia article on PIV provides a good overview of the technique: https://en.wikipedia.org/wiki/Particle_image_velocimetry

1.1 Code description

The library `piv_parallel.py` includes a custom implementation of PIV that is designed to be compatible with X-ray radiographs of flowing fibers. Some key features of this technique include: + Optimized implementation of correlation algorithms, leveraging "fast" Numpy algorithms and matrix slicing + Background correction of images, to correct for time-invariant intensity variation in the radiographs that is not corrected during flat-field correction; these artifacts may occur due to uneven thickness of the nozzle walls. Note that we assume that the iamges are already flat-field corrected, to remove artifacts due to uneven illumination. + Parallel implementation of algorithm using Joblib + Modular implementation, allowing users to change correlation algorithms, etc.

2 Loading images for analysis

To begin, we will inspect certain images. In particular, this analysis requires the following: 1. An input file, describing PIV analysis settings 2. A series of X-ray images. We will calculate displacement between consecutive images (e.g., image_0001 –> image_0002, and image_0002 –> image_0003) 3. A mask image, to define the search locations.

2.0.1 Directory organization

For the current experiment, we studied two variables of interest: 1. Location within nozzle (flow conditions near the tip vs. upstream) 2. Flow velocity (extrusion pressure)

Thus, our files are organized as follows:

```
Main directory:
| PIV_introduction.ipynb (you are here!)
| piv_parallel.py (The main code)
```

```
| "Parent folder" (e.g., Location 1)
| | mask.png
| | "Child folder 1" (contains a series of images)
| | "Child folder 2" (contains a second series of images)
```

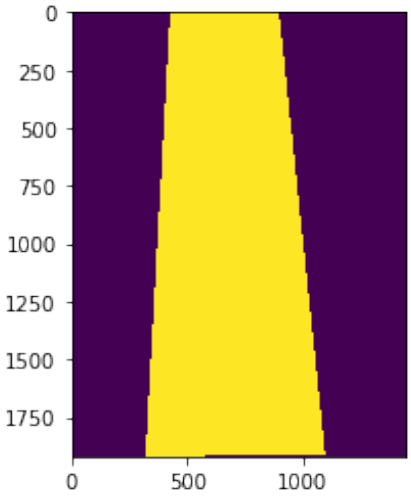
[199]:
```
import os
import numpy as np
import imageio as io
import pandas as pd
from matplotlib import pyplot as plt
import mask_creator as mc

import piv_parallel as piv  # Import the PIV code, stored locally in directory

# Where the files are located:
parent_folder = 'z5.0_nozzle'  # Files located at z=2.5 mm
child_folder = 'sic10to2_movie_z5.0n2_p5.5_1_ANDOR2_001'

image_list = [f for f in os.listdir(os.path.join(parent_folder, child_folder))]

# Define the PIV analysis parameters:
window_halfsize = 80
window_size = 2 * window_halfsize + 1  # Size of the undeformed facet

window_spacing = window_halfsize  # spacing between adjacent subsets

search_halfsize = int(1.5 * window_halfsize)
search_size = 2 * search_halfsize + 1  # How large of an area to correlate?
```

In the cell below, you will create a "mask" that encompasses the area of the flow field. This will give the code a region to measure displacement of pixels between frames so that the velocity can be calculated. * Note: After running the cell, a new window should pop up with an interactive interface. If it prints it below, run the cell a few more times until a new window appears.

[200]:
```
import mask_creator as mc

%matplotlib qt
img = io.imread(os.path.join(parent_folder, child_folder, image_list[1]))

ax = plt.subplot(111)
ax.imshow(img)

mask = mc.MaskCreator(ax)
plt.show()
```

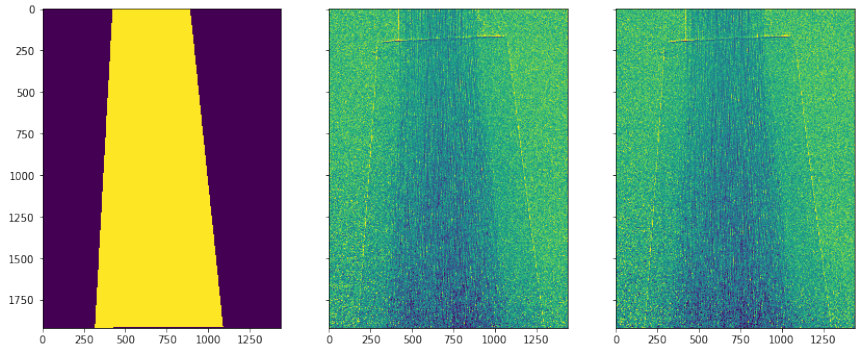
Now run the cell below to verify that the mask you created is an accurate representation of the flow field.

[165]:
```
%matplotlib inline

img_mask = mask.get_mask(img.shape)
plt.imshow(img_mask)
plt.show()

np.save('mask.npy',img_mask)
```



First, we'll load example images to verify the accuracy of the mask. You'll see that the mask neatly identifies the fiber-containing regions of the nozzle. I made this masked image using the free-form polygon tool in Microsoft Paint.

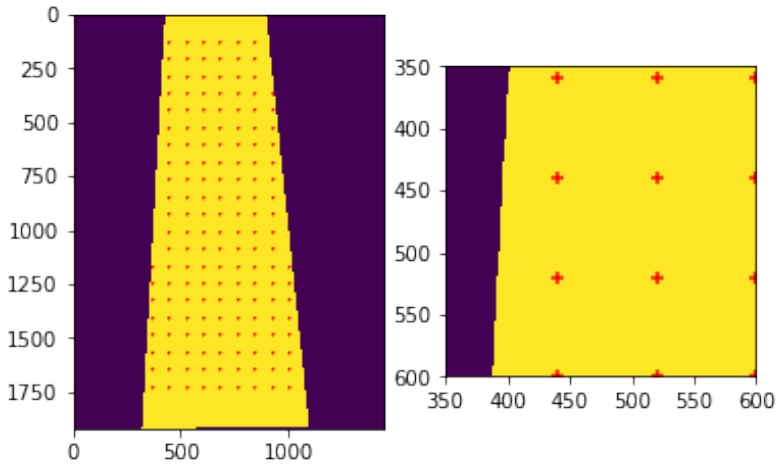
[166]:
```
# Note that "mask.png" has RGBA channels... we only need a binary image
img_mask = np.load('mask.npy')
img_a = io.imread(os.path.join(parent_folder, child_folder, image_list[1]))
img_b = io.imread(os.path.join(parent_folder, child_folder, image_list[2]))

# Plot the images
f, axes = plt.subplots(1, 3, sharex=True, sharey=True, figsize=(14,14))
axes[0].imshow(img_mask)
axes[1].imshow(img_a)
axes[2].imshow(img_b)
plt.show()
```



2.1 Identifying points for PIV analysis

Next, we'll generate a list of subsets to be analyzed from the mask image. The method `piv.build_grid_points_from_mask()` makes a periodic grid of points that fall inside the masked region.

[194]:
```
xy_search = piv.build_grid_points_from_mask(
    img_mask, window_spacing, search_halfsize
)

# Plot on top of the masked image:
f, axes = plt.subplots(1, 2)
axes[0].imshow(img_mask)
axes[0].scatter(*xy_search.T[::-1], c='r', marker='+', s=1, lw=1)

axes[1].imshow(img_mask)
axes[1].scatter(*xy_search.T[::-1], c='r', marker='+')
axes[1].set_xlim(350, 600); axes[1].set_ylim(350, 600)
axes[1].invert_yaxis()

plt.show()
```



2.2 Correlation of a single subset

Ultimately, we will calculate the displacement using PIV at each marked subset location. This is implemented in the `piv.piv_image_pair()` method. Before that, we will explore the contents of this method.

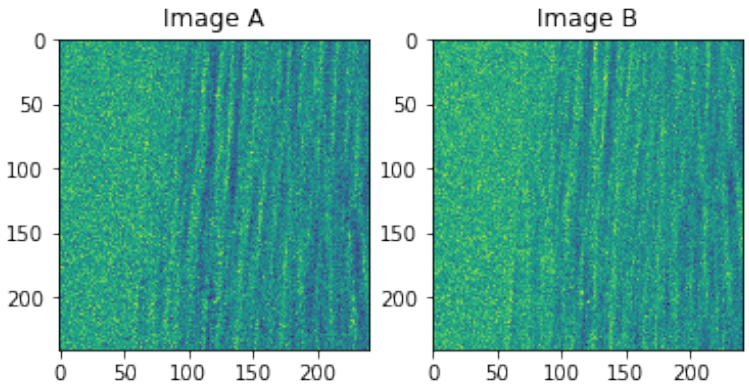
Inspection of `piv.piv_image_pair()` reveals a workflow that looks something like this: 1. Crop `img_a` to create `template` 2. Crop `img_b` to define `search_region` 3. Perform `piv_search()` to compute the displacement between these images. 4. Save the displacement. 5. Update the initial guess, if necessary.

We will demonstrate this procedure on a single subset in the image:

[195]:
```
# Select an arbitrary location to analyze. This subset is located somewhere near␣
↪the center of the image.
i = -100
print('Search coordinates: ', xy_search[i])
template = piv.crop_img(img_a, xy_search[i], window_size)
search_region = piv.crop_img(img_b, xy_search[i], search_size)

# Calculate the displacement. Note that we calculate sub-pixel displacement by␣
↪interpolating the correlation peak.
displacement_i = piv.piv_search(template, search_region)
print('Measured displacement: ', displacement_i)

# Plot the template and facet, to show the small motion between images.
f, axes = plt.subplots(1, 2)
```

```
axes[0].imshow(piv.crop_img(img_a, xy_search[i], search_size))
axes[1].imshow(search_region)
axes[0].set_title('Image A'), axes[1].set_title('Image B')
plt.show()
```

```
Search coordinates:  [840. 440.]
Measured displacement:  [28.98481031 -0.83256099]
```



These results show a small dispalcement of ~3 pixels in the vertical direction.

2.3 PIV analysis of a full image pair

Next, we can repeat this over all regions of the image, using `piv.piv_image_pair()`. Note that we add a small initial guess, which improves the measurement quality a little bit, particularly for large displacements / fast moving flows. This is because we only perform a local search to determine the displacement, rather than a global search.

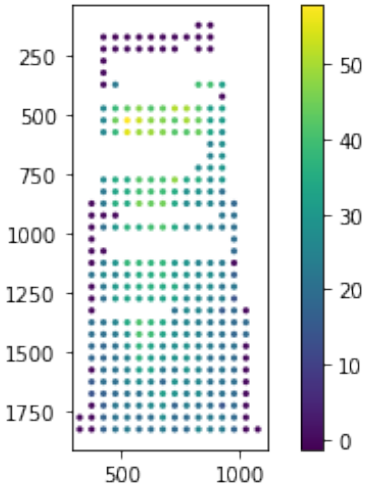
WARNING: This takes about 90 seconds on my computer to perform all calculations. So tread carefully :)

[191]:
```
import time
t1 = time.time()
displacement_array = piv.piv_image_pair(
    img_a, img_b, xy_search, window_size, search_size,
    initial_guess=np.array([5, 0]))
t2 = time.time()
print('{:02.1f} s to complete'.format(t2 - t1))
```

```
38.0 s to complete
```

Visualize the measured displacements using Matplotlib:

[192]:
```
plt.scatter(*xy_search[:, ::-1].T, c=displacement_array[:, 0], lw=0, s=10)
plt.gca().invert_yaxis()
plt.gca().set_aspect('equal')
plt.colorbar()
plt.show()
```



2.4 Summary

These results are relatively good, and closely resemble a theoretical solution for fluid flow though a tapered pipe. In practice, however, you should average the results across multiple images to improve the measurement quality.

This is implemented in `piv.piv_folder()`, which is constructed to enable massively-parallel PIV analysis. An efficient call signature would look like this:

```
from joblib import Parallel, delayed

file_template = 'image_{:04d}.tiff'

# Average all images in 'folder' to remove image artifacts:
avg_img = piv.average_images_in_folder(folder, batch_size=8)

def main(img_i):
            piv.piv_folder(
                folder, outfolder, file_template, img_i, facet_size, search_size,
                img_step=img_step, initial_guess=initial_guess,
                avg_img=avg_img
```

```
                )

# Run on all available cores
Parallel(n_jobs=-1)(delayed(main)(x) for x in range(0, 99, 1))
```

This outputs a separate CSV for each image. These results can subsequently be combined using `piv.average_PIV_results()`

Finally, note that the code is decently documented. More information is available by calling `help`.

[7]:
```
help(piv.piv_folder)
```

```
Help on function piv_folder in module piv_parallel:

piv_folder(folder, outfolder, file_template, i, window_halfsize,
search_halfsize, img_step=1, initial_guess=array([0, 0]), avg_img=None)
    Correlate a pair of images located in "folder" in a way that's amenable to
    massively-parallel computation.

    Inputs:
    + folder: where the images are located
    + outfolder: where to save the PIV results
    + file_template: string that is used to identify folder.
    + i: ID of the "reference" image, i.e., file_template.format(i)
    + img_step: "deformed" image = i + img_step
    + window_halfsize, search_halfsize: Dimensions of the facet, and the search
region
    + initial_guess: An estimate to narrow down the search region

    Edited: 4 JUN 2020
```