
X-ray Image Analysis Documentation

Release 0.0.1

Argonne National Laboratory

Jul 28, 2017

Contents

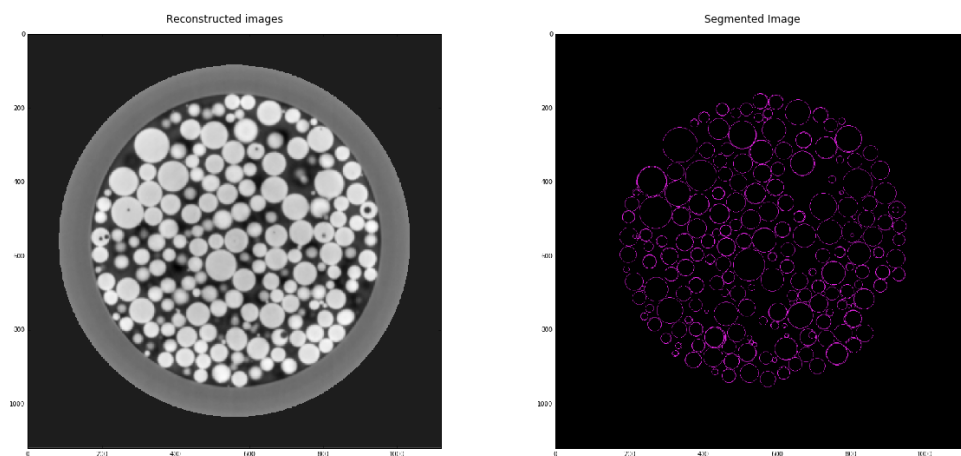
1	Few examples	3
2	How to Contribute	5
	Bibliography	29
	Python Module Index	31

The X-image is a collection of tools and tutorials for segmenting X-ray and RGB images using image processing and compute vision techniques.

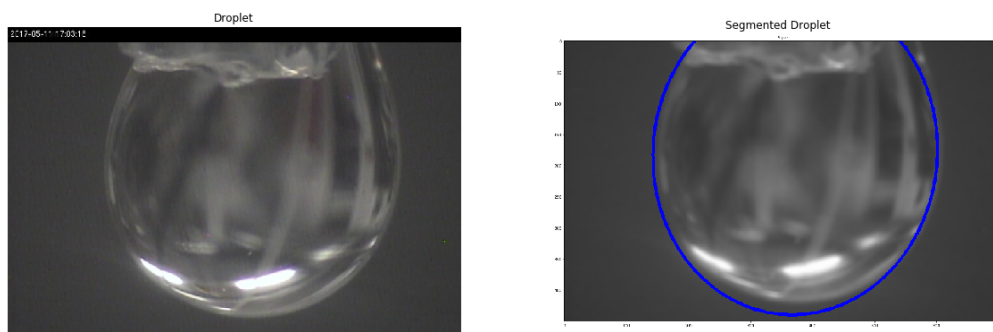
CHAPTER 1

Few examples

Detecting location and size of cells in X-ray images:



Detecting droplets using RGB camera images:



CHAPTER 2

How to Contribute

If you are working on a segmentation tool and would like to contribute

- Documentation: <https://github.com/tomography/ximage/tree/master/doc>
- Issue Tracker: <https://github.com/tomography/ximage/docs/issues>
- Source Code: <https://github.com/tomography/ximage/ximage>

About

The **X-Image** is a collaborative space for gathering tools and tutorials for segmenting data for X-Ray sciences.

Install

This section covers the basics of how to download and install **X-Image**. We recommend you to install the **Anaconda Python** distribution.

Contents:

- *Installing from source*

Installing from source

Clone **ximage** from **GitHub** repository:

```
git clone https://github.com/tomography/ximage.git ximage
```

then:

```
cd ximage
python setup.py install
```

Development

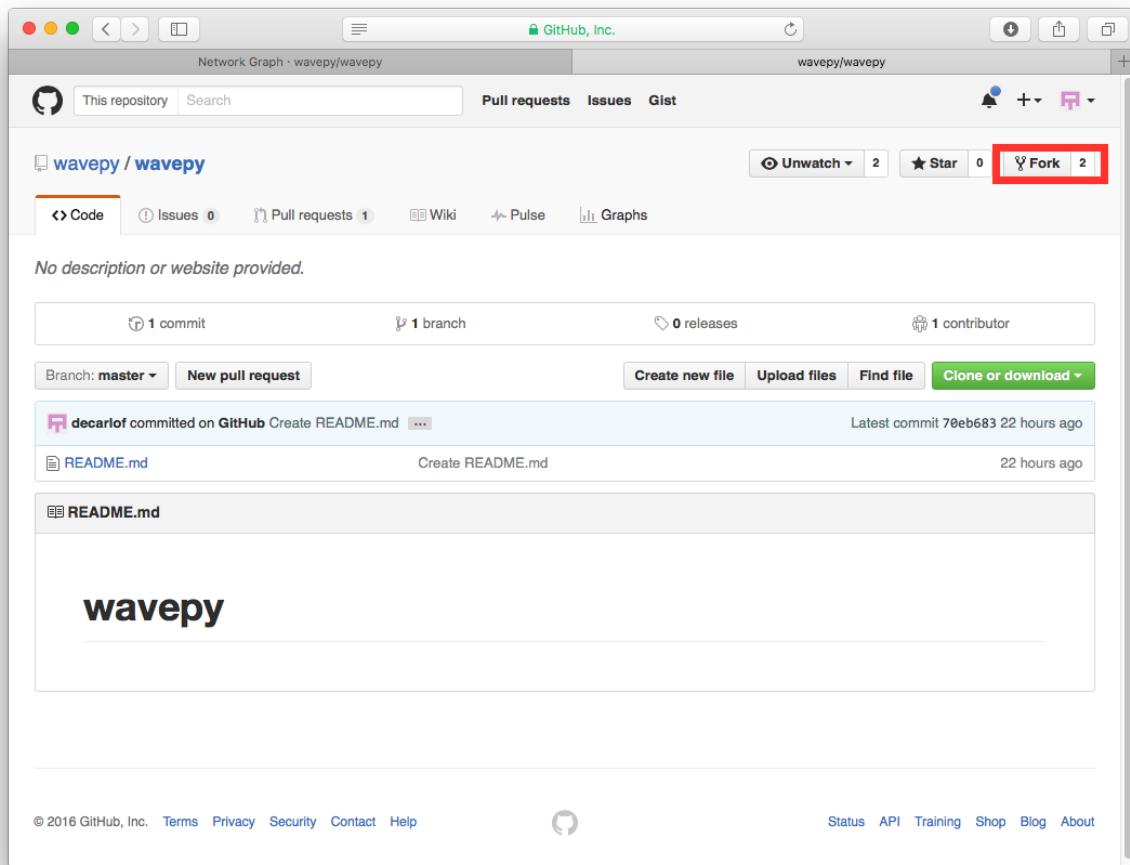
This section explains the basics for developers who wish to contribute to a project maintained on GitHub and using the [fork / pull request](#) mechanism for accepting developer contributions.

Contents:

- *Fork the repository*
- *Clone the repository*
- *Coding conventions*
- *Package versioning*
- *Committing changes*
- *Contributing back*

Fork the repository

The project is maintained on GitHub, which is a version control and a collaboration platform for software developers. To start first register on [GitHub](#) and fork the [Project repository](#) by clicking the **Fork** button in the header of the [Project repository](#):

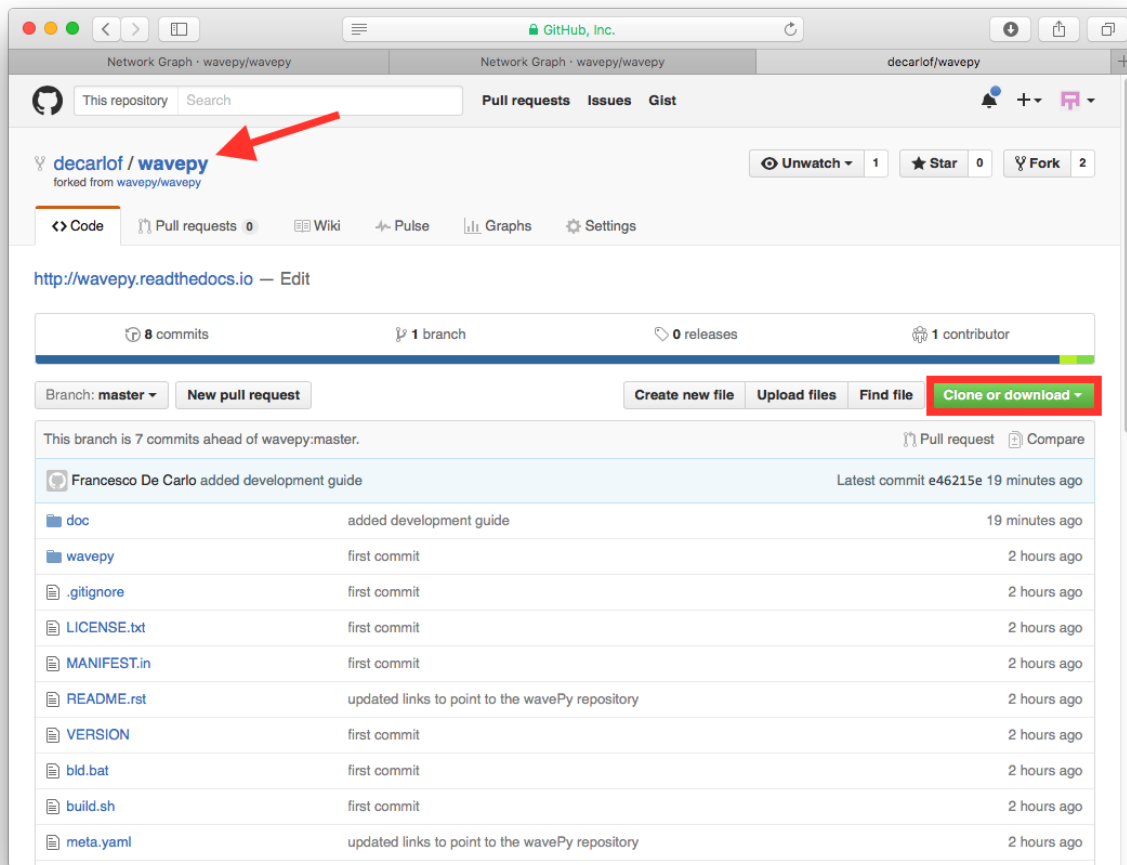


This successfully creates a copy of hspeed in your personal GitHub space.

Clone the repository

The next thing you want to do is to clone the repository you just created in your personal GitHub space to your local machine.

You can do this by clicking the **Clone in Desktop** button in the bottom of the right hand side bar:



This will launch the GitHub desktop application (available for both [Mac](#) and [Win](#)) and ask you where you want to save it. Select a location in your computer and feel comfortable with making modifications in the code.

Coding conventions

We try to keep a consistent and readable code. So, please keep in mind the following style and syntax guidance before you start coding.

First of all the code should be well documented, easy to understand, and integrate well into the rest of hspeed. For example, when you are writing a new function always describe the purpose and the parameters:

```
def my_awesome_func(a, b):  
    """  
    Adds two numbers.  
  
    Parameters  
    -----  
    a : scalar (float)  
        First number to add  
  
    b : scalar (float)  
        Second number to add
```

```

Returns
-----
output : scalar (float)
    Added value
"""
return a+b

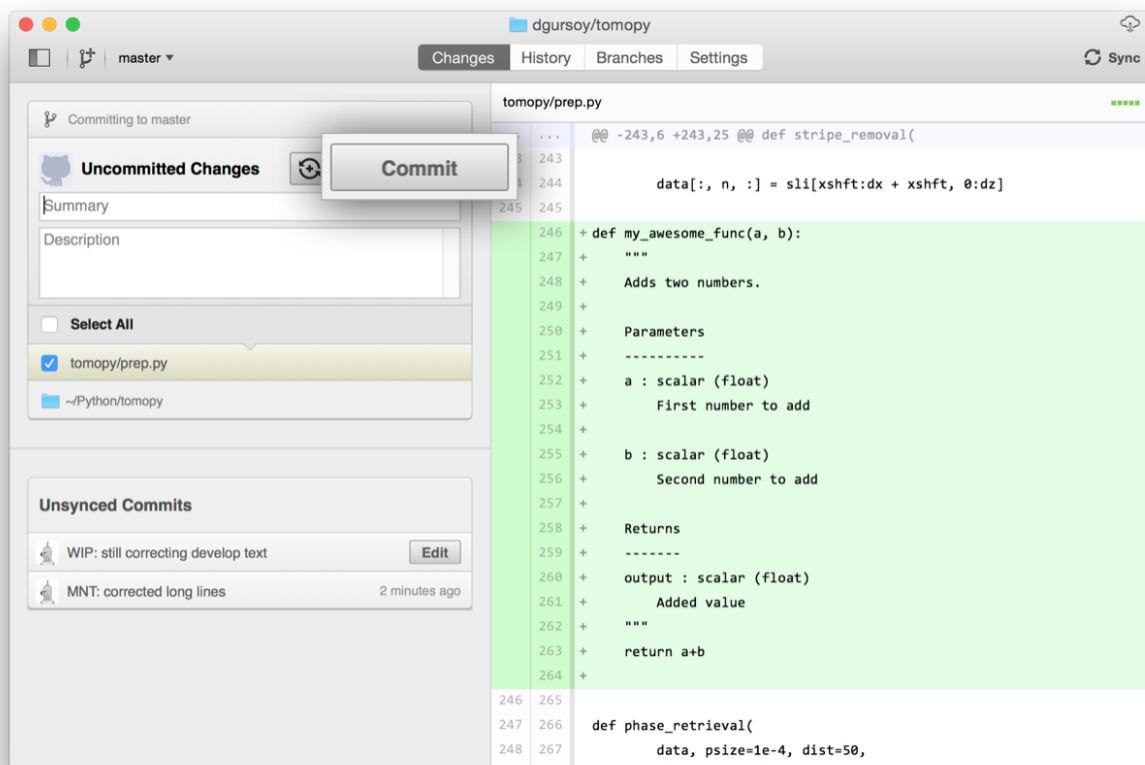
```

Package versioning

We follow the X.Y.Z (Major.Minor.Patch) semantic for package versioning. The version should be updated before each pull request accordingly. The patch number is incremented for minor changes and bug fixes which do not change the software's API. The minor version is incremented for releases which add new, but backward-compatible, API features, and the major version is incremented for API changes which are not backward-compatible. For example, software which relies on version 2.1.5 of an API is compatible with version 2.2.3, but not necessarily with 3.2.4.

Committing changes

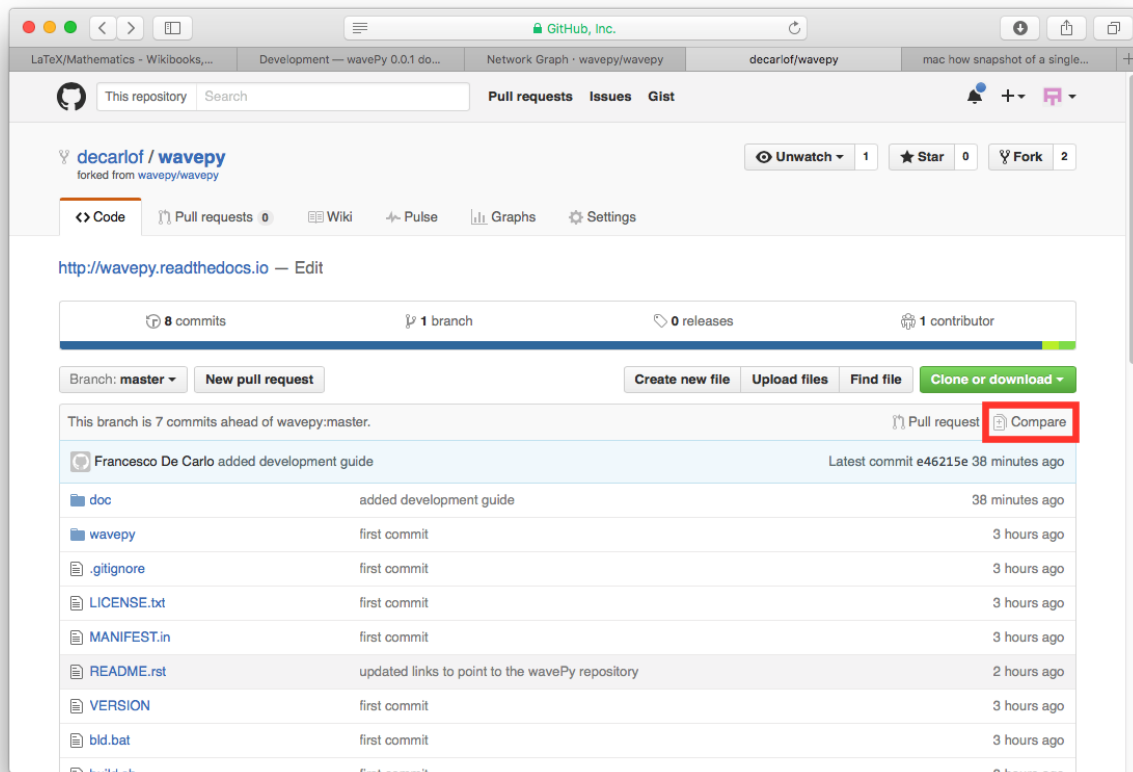
After making some changes in the code, you may want to take a *snapshot* of the edits you made. That's when you make a *commit*. To do this, launch the GitHub desktop application and it should provide you all the changes in your code since your last commit. Write a brief *Summary* and *Description* about the changes you made and click the **Commit** button:



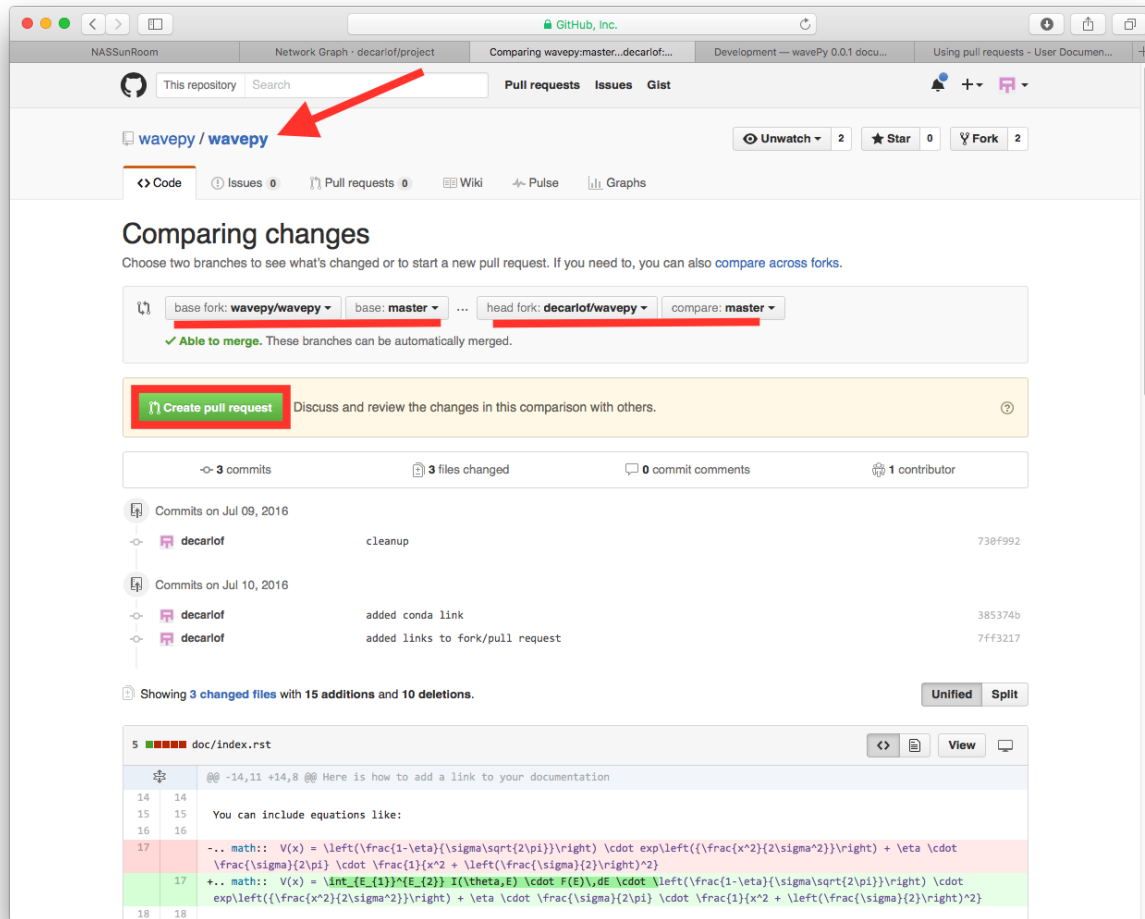
You can continue to make changes, add modules, write your own functions, and take more *Commit snapshots* of your code writing process.

Contributing back

Once you feel that the functionality you added would benefit the community, then you should consider contributing back to hspeed. For this, go to your online GitHub repository of hspeed and click on the *compare* button to compare, review and create a pull request.



After clicking on this button, you are presented with a review page where you can get a high-level overview of what exactly has changed between your forked branch and the original project repository. When you're ready to submit your pull request, click **Create pull request**:



Clicking on **Create pull request** sends you to a discussion page, where you can enter a title and optional description. It's important to provide as much useful information and a rationale for why you're making this Pull Request in the first place.

When you're ready typing out your heartfelt argument, click on **Send pull request**.

You're done!

API reference

ximage Modules:

`ximage.util`

Utility module for ximage

Functions:

<code>load_raw(top, index_start)</code>	Load a stack of tiff images.
<code>shutter_off(image[, alpha])</code>	Finds the first image with the shutter closed.
<code>particle_bed_location(image)</code>	Finds the image row marking the location of the particle bed.
<code>laser_on(rdata, particle_bed_ref[, alpha])</code>	Function description.
<code>scale_to_one(ndata)</code>	Scale a stack of images between [0,1].
<code>sobel_stack(ndata)</code>	Applies sobel filter to a stack of images.
<code>label(ndata[, blur_radius, alpha])</code>	Counts the number of particles in a stack of images.

`ximage.util.load_raw(top, index_start)`

Load a stack of tiff images.

Parameters

- **top** (*str*) – Top data directory.
- **index_start** (*int*) – Image index start.

Returns *ndarray* – 3D stack of images.

`ximage.util.shutter_off(image, alpha=0.7)`

Finds the first image with the shutter closed.

Parameters

- **image** (*ndarray*) – 3D stack of images.
- **alpha** (*float*) – Threshold level.

Returns *int* – Index of the first image with the shutter closed.

`ximage.util.particle_bed_location(image)`

Finds the image row marking the location of the particle bed.

Parameters **image** (*ndarray*) – 2D image.

Returns *int* – Image row marking the location of the particle bed.

`ximage.util.laser_on(rdata, particle_bed_ref, alpha=1.0)`

Function description.

Parameters

- **rdata** (*ndarray*) – 3D stack of images.
- **particle_bed_ref** (*int*) – Image row marking the location of the particle bed.
- **alpha** (*float*) – Threshold level.

Returns *int* – Index of the first image with the laser on.

`ximage.util.scale_to_one(ndata)`

Scale a stack of images between [0,1].

Parameters **ndata** (*ndarray*) – 3D stack of images.

Returns *ndarray* – 3D stack of images.

`ximage.util.sobel_stack(ndata)`

Applies sobel filter to a stack of images.

Parameters **ndata** (*ndarray*) – 3D stack of images.

Returns *ndarray* – 3D stack of images.

`ximage.util.label` (*ndata*, *blur_radius=1.0*, *alpha=1*)

Counts the number of particles in a stack of images.

Parameters

- **ndata** (*ndarray*) – 3D stack of images.
- **blur_radius** (*float*) – Gaussian blur radius.
- **alpha** (*float*) – Threshold level.

Returns *ndata*, *nr_objects* – 3D stack of images, number of particles per image.

`ximage.widget`

This module contains the ximage widgets

Functions:

<code>slider</code> (<i>data</i>)	Plots a stack of images.
-------------------------------------	--------------------------

class `ximage.widget.slider` (*data*)

Plots a stack of images.

Parameters **data** (*ndarray*) – 3D stack of images.

update (*val*)

Examples

This section contains [Jupyter Notebooks](#) and Python scripts examples for various ximage functions.

To [run](#) these examples in a [notebooks](#) install [Jupyter](#) or run the python scripts from [here](#)

Particle Tracking

This is an example of particle tracking in additive manufacturing high speed x-ray imaging. You can download this example as `jupyter notebook` or as `python script`

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import ximage
import fnmatch
import tomopy
import numpy as np
```

```
top = '/local/dataraid/am/104_Ti_04_p90_S1/'
```

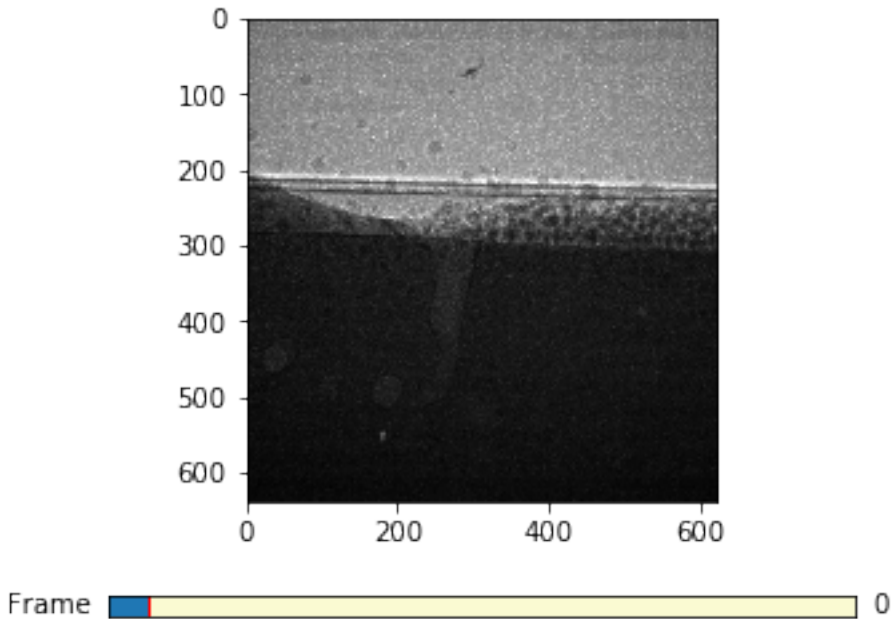
```
index_start = 1
```

```
# Total number of images to read
nfile = len(fnmatch.filter(os.listdir(top), '*.tif'))
print(nfile)
```

```
361
```

```
rdata = ximage.load_raw(top, index_start)
```

```
ximage.slider(rdata[150:160,:,:])
```



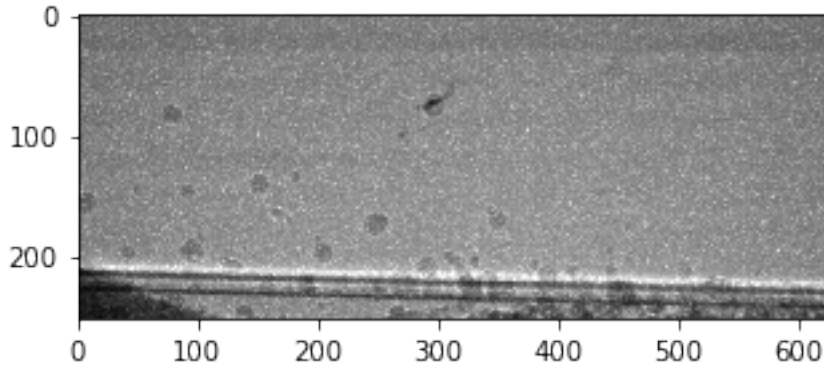
```
particle_bed_reference = ximage.particle_bed_location(rdata[0])
```

```
print("Particle bed location: ", particle_bed_reference)
```

```
('Particle bed location: ', 253)
```

```
# Cut the images to remove the particle bed
cdata = rdata[:, 0:particle_bed_reference, :]
```

```
ximage.slider(cdata[150:160,:,:])
```



Frame  0

```
# Find the image when the shutter starts to close
dark_index = ximage.shutter_off(rdata)
print("Shutter CLOSED on image: ", dark_index)
```

```
('Shutter CLOSED on image: ', 344)
```

```
# Find the images when the laser is on
laser_on_index = ximage.laser_on(rdata, particle_bed_reference, alpha=1.0)
print("Laser ON on image: ", laser_on_index)
```

```
('Laser ON on image: ', 46)
```

```
# Set the [start, end] index of the blocked images, flat and dark.
flat_range = [0, 1]
data_range = [laser_on_index, dark_index]
dark_range = [dark_index, nfile]

flat = cdata[flat_range[0]:flat_range[1], :, :]
proj = cdata[data_range[0]:data_range[1], :, :]
dark = np.zeros((dark_range[1]-dark_range[0], proj.shape[1], proj.shape[2]))
```

```
# Normalize the images
ndata = tomopy.normalize(proj, flat, dark)
ndata = tomopy.normalize_bg(ndata, air=ndata.shape[2]/2.5)
ndata = tomopy.minus_log(ndata)
```

```
blur_radius = 3.0
threshold = .04
nndata = ximage.label(ndata, blur_radius, threshold)
```

```
Image 0 contains 8 particles
Image 1 contains 13 particles
Image 2 contains 10 particles
Image 3 contains 13 particles
Image 4 contains 12 particles
Image 5 contains 19 particles
```

```
Image 6 contains 21 particles
Image 7 contains 26 particles
Image 8 contains 23 particles
Image 9 contains 22 particles
Image 10 contains 24 particles
Image 11 contains 25 particles
Image 12 contains 29 particles
Image 13 contains 28 particles
Image 14 contains 29 particles
Image 15 contains 26 particles
Image 16 contains 28 particles
Image 17 contains 24 particles
Image 18 contains 24 particles
Image 19 contains 21 particles
Image 20 contains 23 particles
Image 21 contains 24 particles
Image 22 contains 23 particles
....
Image 295 contains 61 particles
Image 296 contains 57 particles
Image 297 contains 55 particles
```

Circle Detection

Here is an example of a tomographic data set of a sample consisting of spheres. The goal is to detect the circles contained in a tomographic reconstructed slice. First step is the tomographic reconstruction followed by sharpening, defect detection and finally circle detection.

You can download this example as `jupyter notebook`

```
import matplotlib.pyplot as plt
%matplotlib inline
```

Image reconstruction

```
import tomopy
import dxchange

# Set path to the micro-CT data to reconstruct.
fname = '/local/decarlo/sector1/g120f5/g120f5_'

# Select the sinogram range to reconstruct.
start = 100
end = 116

# Read the APS 1-ID raw data.
proj, flat, dark = dxchange.read_aps_lid(fname, sino=(start, end))

# Set data collection angles as equally spaced between 0-180 degrees.
theta = tomopy.angles(proj.shape[0], angl=0.0, ang2=360.0)

# Flat-field correction of raw data.
proj = tomopy.normalize(proj, flat, dark)

# Ring removal.
```

```
proj = tomopy.remove_stripe_sf(proj)

# phase retrieval
proj = tomopy.retrieve_phase(proj, alpha=1e-3, pad=True)

# -log.
proj = tomopy.minus_log(proj)

# Reconstruct object using Gridrec algorithm.
rec = tomopy.recon(proj, theta, center=576, algorithm='gridrec')

# Mask each reconstructed slice with a circle.
rec = tomopy.circ_mask(rec, axis=0, ratio=0.85)
```

```
plt.figure(figsize=(12, 12))
plt.imshow(rec[8], cmap='gray', interpolation='none')
```

```
<matplotlib.image.AxesImage at 0x7fecb2578ef0>
```

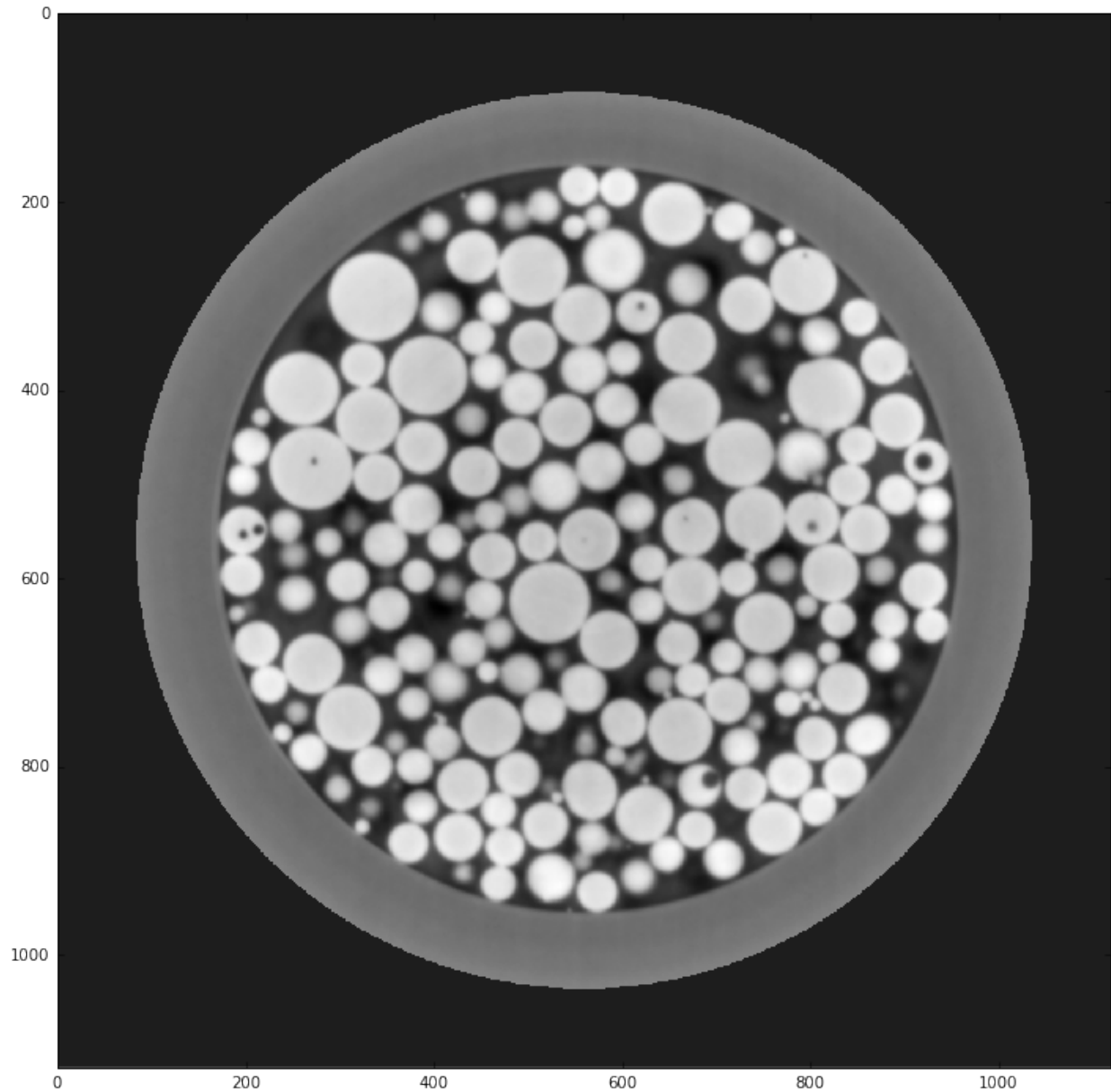


Image sharpening

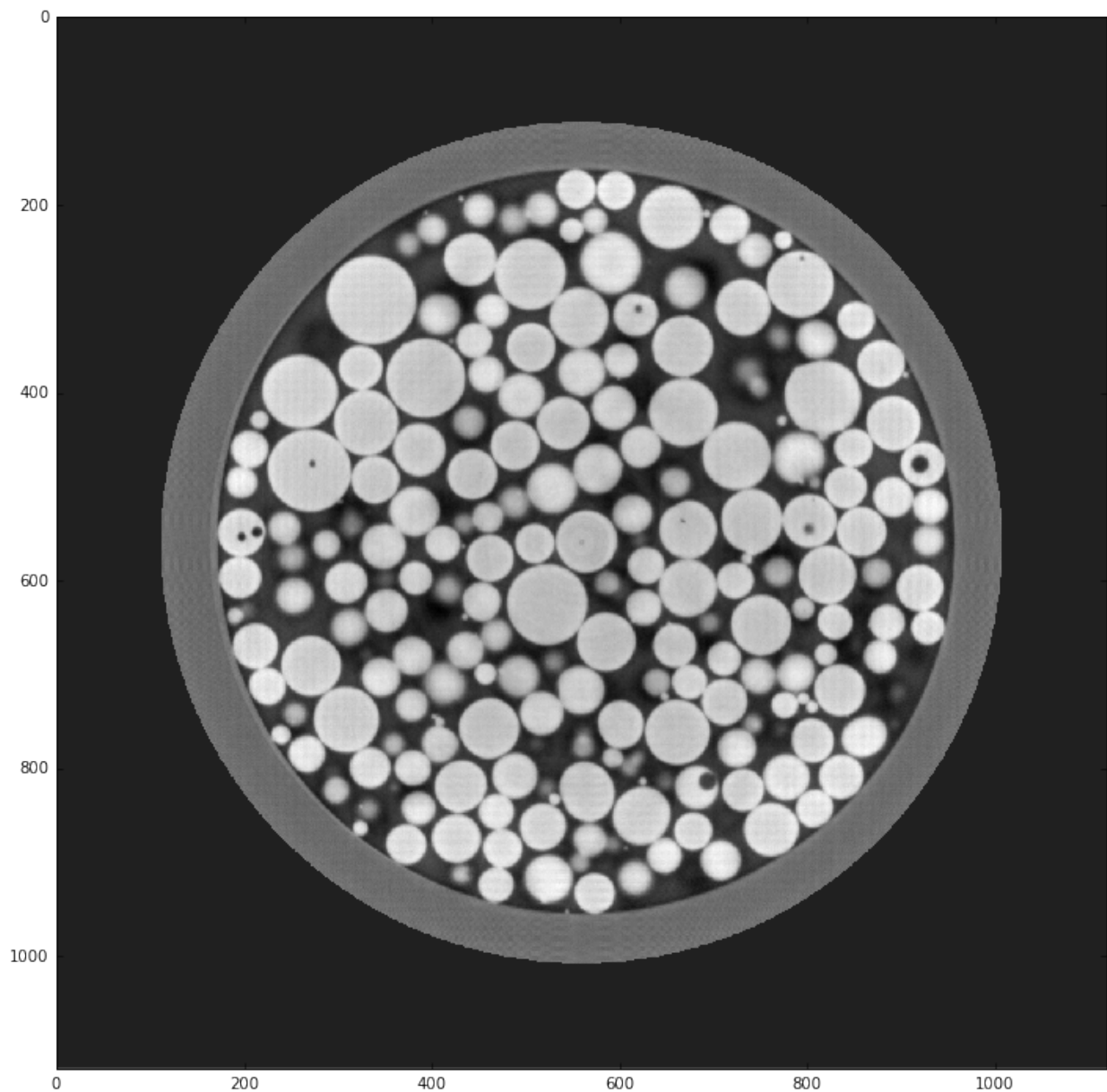
```
import cv2
import numpy as np

# Pick one slice for further processing.
img = rec[8].copy()

kernel = np.array([[ -1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
img = cv2.filter2D(img, -1, kernel)
img = tomopy.circ_mask(np.expand_dims(img, 0), axis=0, ratio=0.8)[0]

plt.figure(figsize=(12, 12))
plt.imshow(img, cmap='gray', interpolation='none')
```

```
<matplotlib.image.AxesImage at 0x7fec14e48048>
```



Artifact detection

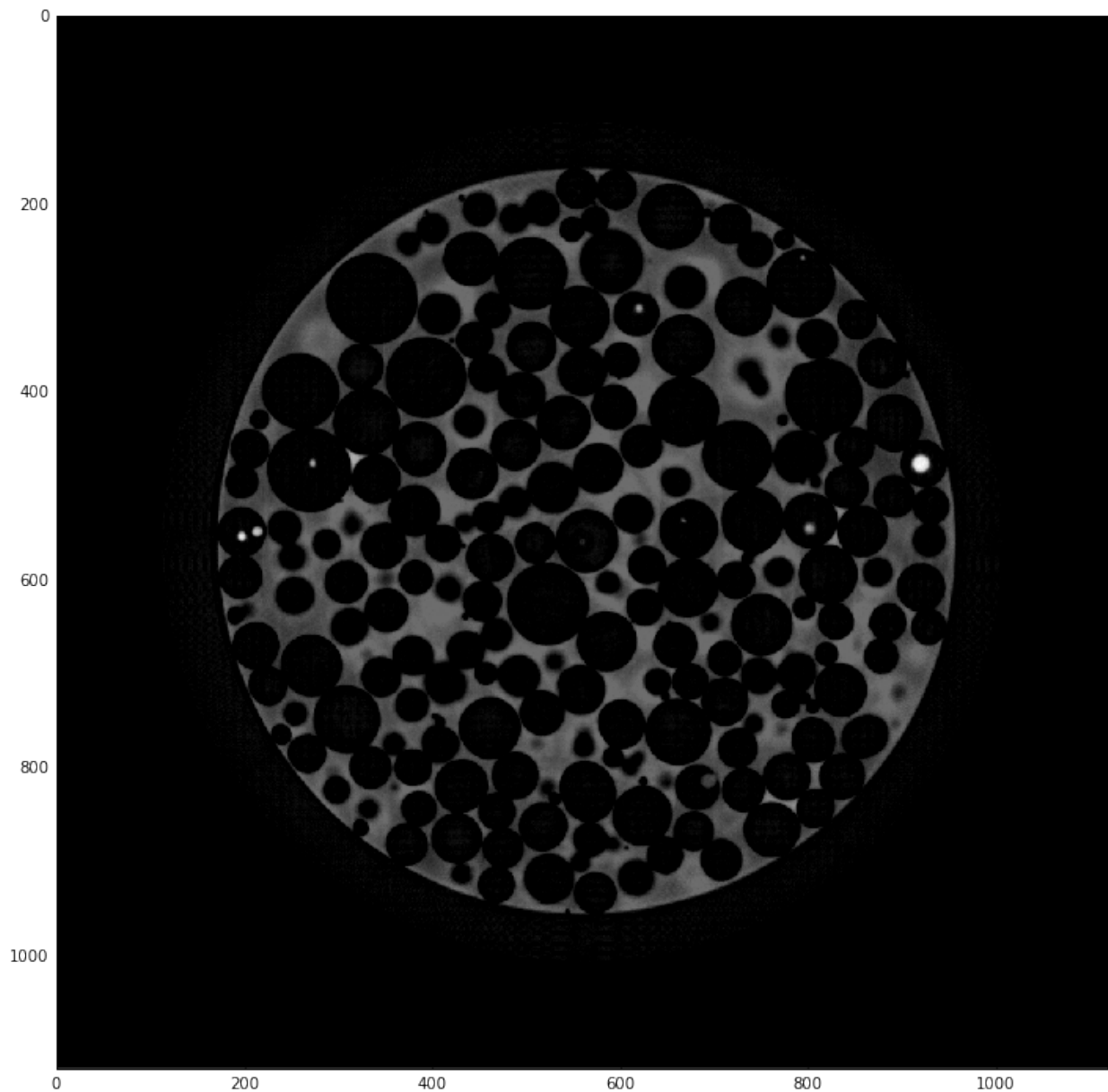
```
from skimage.morphology import reconstruction

img0 = (255 * (img - img.min()) / (img - img.min()).max()).astype('uint8')
mask = img0
seed1 = np.copy(img0)
seed2 = np.copy(img0)
seed1[1:-1, 1:-1] = img0.max()
seed2[1:-1, 1:-1] = img0.min()
eris = reconstruction(seed1, mask, method='erosion')
```

```
dila = reconstruction(seed2, mask, method='dilation')
img0 = (eris+dila-img0)
# img0 = img0 > 120

plt.figure(figsize=(12, 12))
plt.imshow(img0, cmap='gray', interpolation='none')
```

```
<matplotlib.image.AxesImage at 0x7fec141e9a20>
```



Circle detection

```
from skimage import color
from skimage.transform import hough_circle, hough_circle_peaks
```



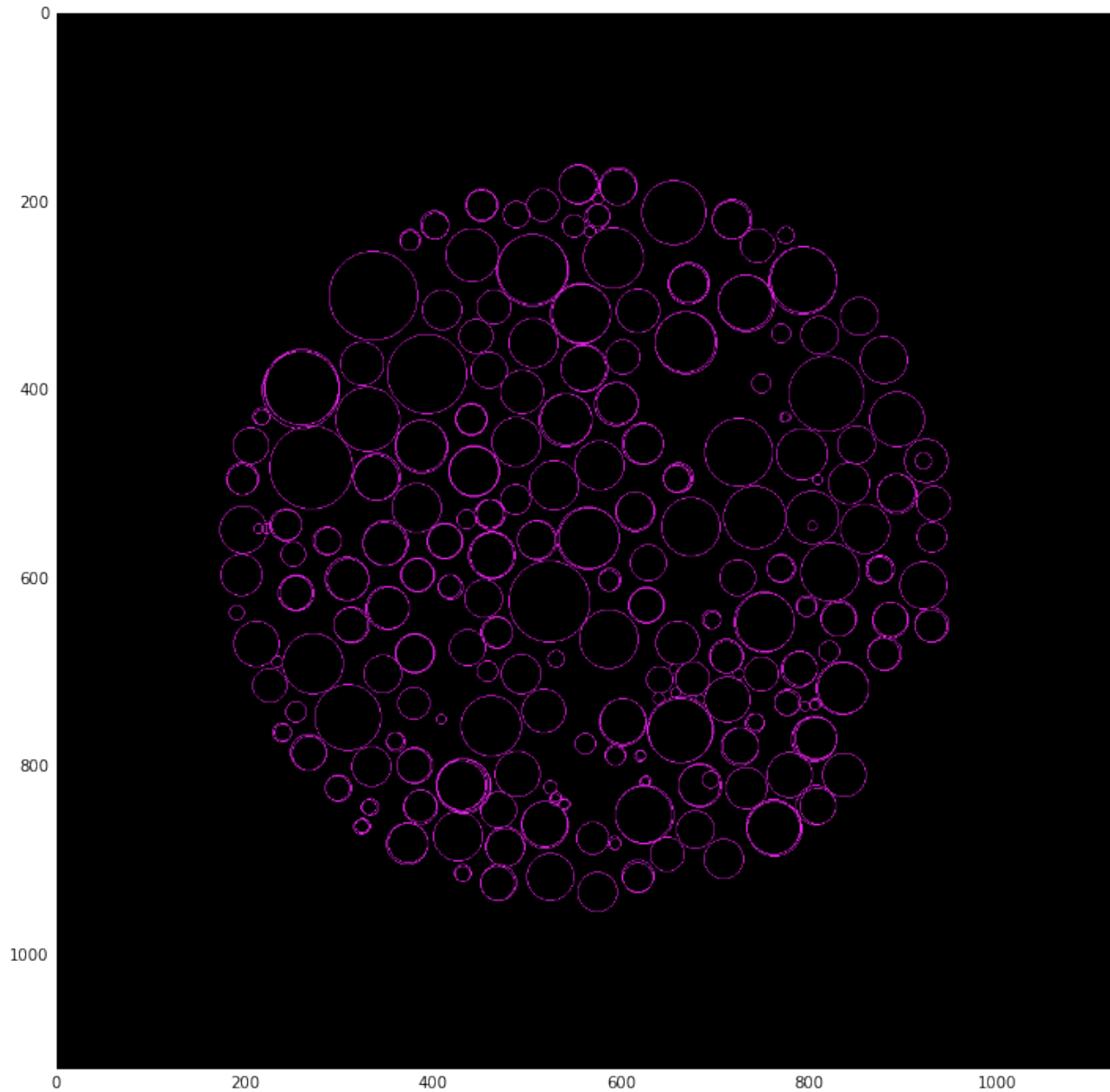
```
from skimage.feature import canny
from skimage.draw import circle_perimeter

img0 = (255 * (img - img.min()) / (img - img.min()).max()).astype('uint8')
edges = canny(img0, sigma=2)
hough_radii = np.arange(5, 50, 1)
hough_res = hough_circle(edges, hough_radii)
accums, cx, cy, radii = hough_circle_peaks(hough_res, hough_radii, total_num_
↪peaks=300)

img1 = np.zeros(img0.shape)
img1 = color.gray2rgb(img1)
for center_y, center_x, radius in zip(cy, cx, radii):
    circy, circx = circle_perimeter(center_y, center_x, radius)
    img1[circy, circx] = (20, 220, 20)

plt.figure(figsize=(12, 12))
plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7fec13d65908>
```



Droplet Detection

You can download this example as `jupyter notebook`

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import glob

from ipywidgets import interact

get_ipython().magic('matplotlib inline')
plt.rcParams['figure.figsize'] = (20.0, 15.0) # set default size of plots
```

```
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
plt.rcParams['axes.titlesize'] = 8

%matplotlib inline
```

Detecting Background Pixels

```
bg = cv2.imread('./samples/bg_nolbl.jpg', 0)
droplet = cv2.imread('./samples/full_droplet_1.jpg', 0)

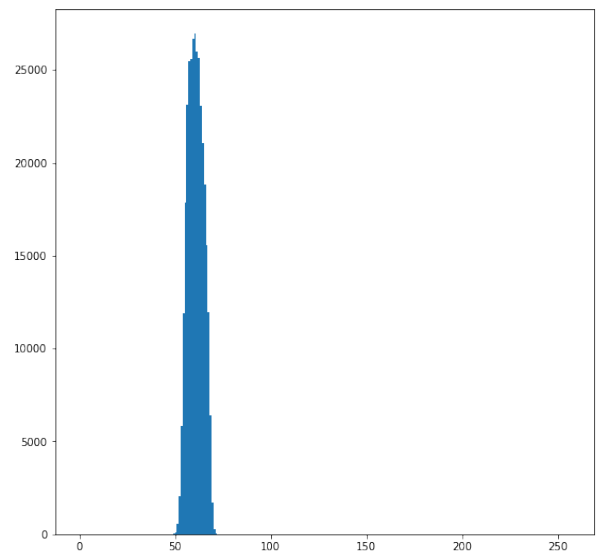
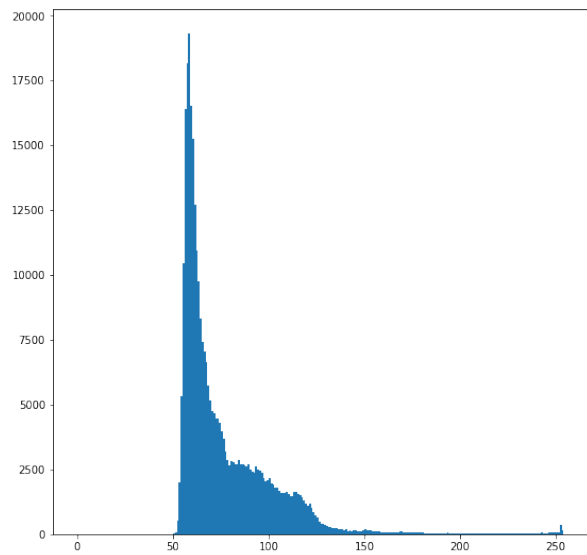
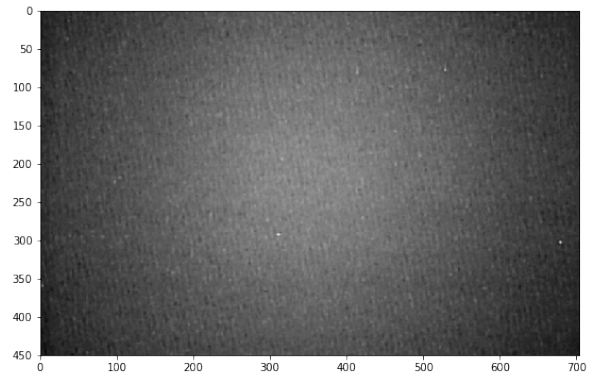
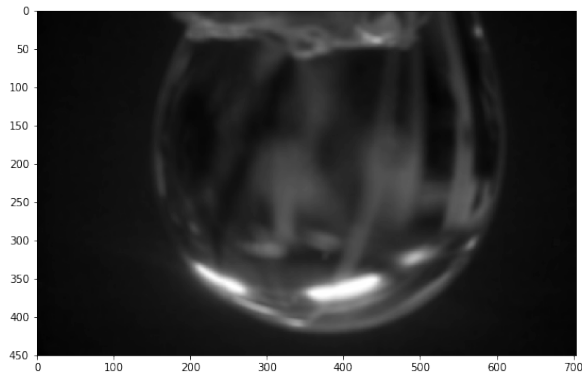
# crop the top
bg = bg[30:, :]
droplet = droplet[30:, :]

droplet = cv2.blur(droplet, ksize=(9,9))
bg = cv2.blur(bg, ksize=(3,3))

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(20, 20))

ax1.imshow(droplet)
ax2.imshow(bg)
ax3.hist(droplet.ravel(), 256, [0, 256])
ax4.hist(bg.ravel(), 256, [0, 256])

plt.show()
```

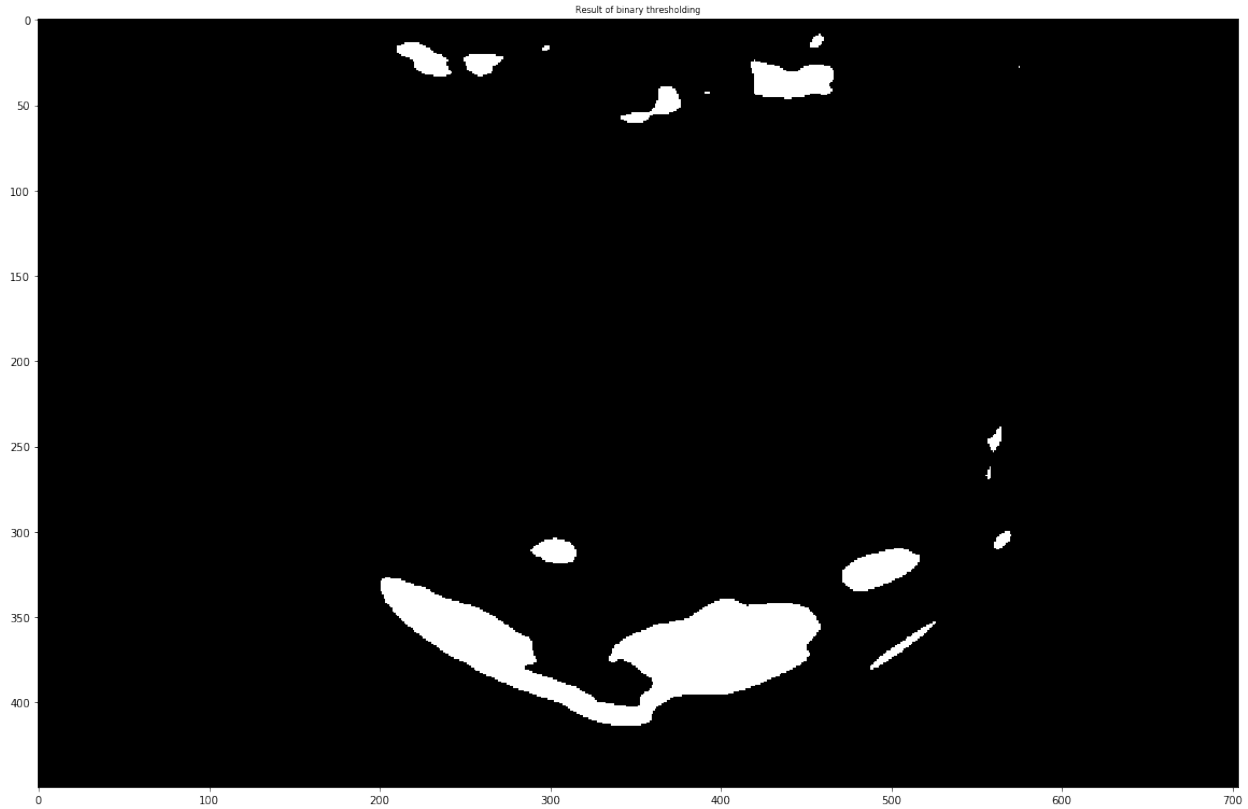


Apply thresholds

```
low_thresh = 100
high_thresh = 250

def apply_thresh(low, high):
    ret, binary = cv2.threshold(droplet, low, high, 0)
    plt.figure(figsize=(20, 20))
    plt.imshow(binary, cmap='gray')
    plt.title('Result of binary thresholding')
    plt.show()

interact(apply_thresh, low=(0, 250), high=(0, 255));
```



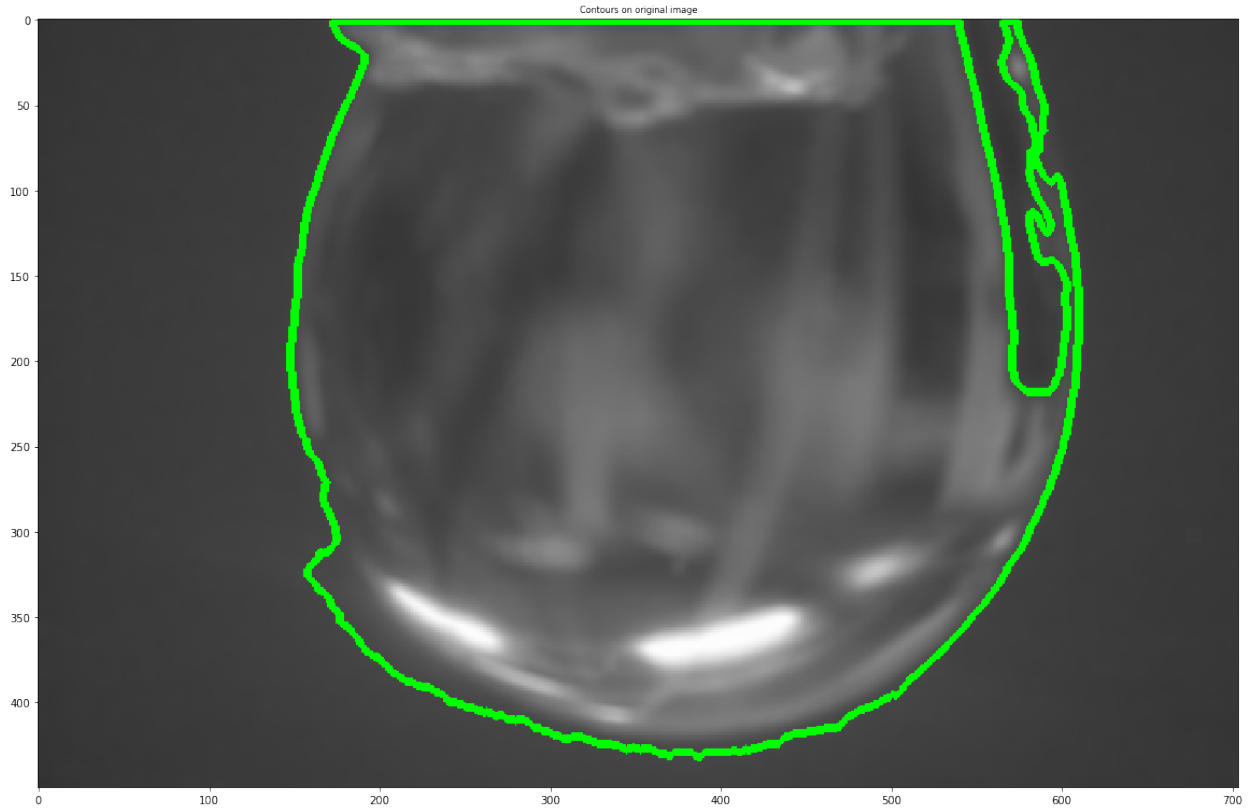
Finding Contours

```
low_thresh = 70
high_thresh = 255

ret, binary = cv2.threshold(droplet, low_thresh, high_thresh, 0)

im2, contours, hierarchy = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_
↪APPROX_SIMPLE)
segmented_img = np.dstack((droplet, droplet, droplet))
cv2.drawContours(segmented_img, contours, -1, (0, 255, 0), 3)

plt.figure(figsize=(20, 20))
plt.title('Contours on original image')
plt.imshow(segmented_img)
plt.show()
```

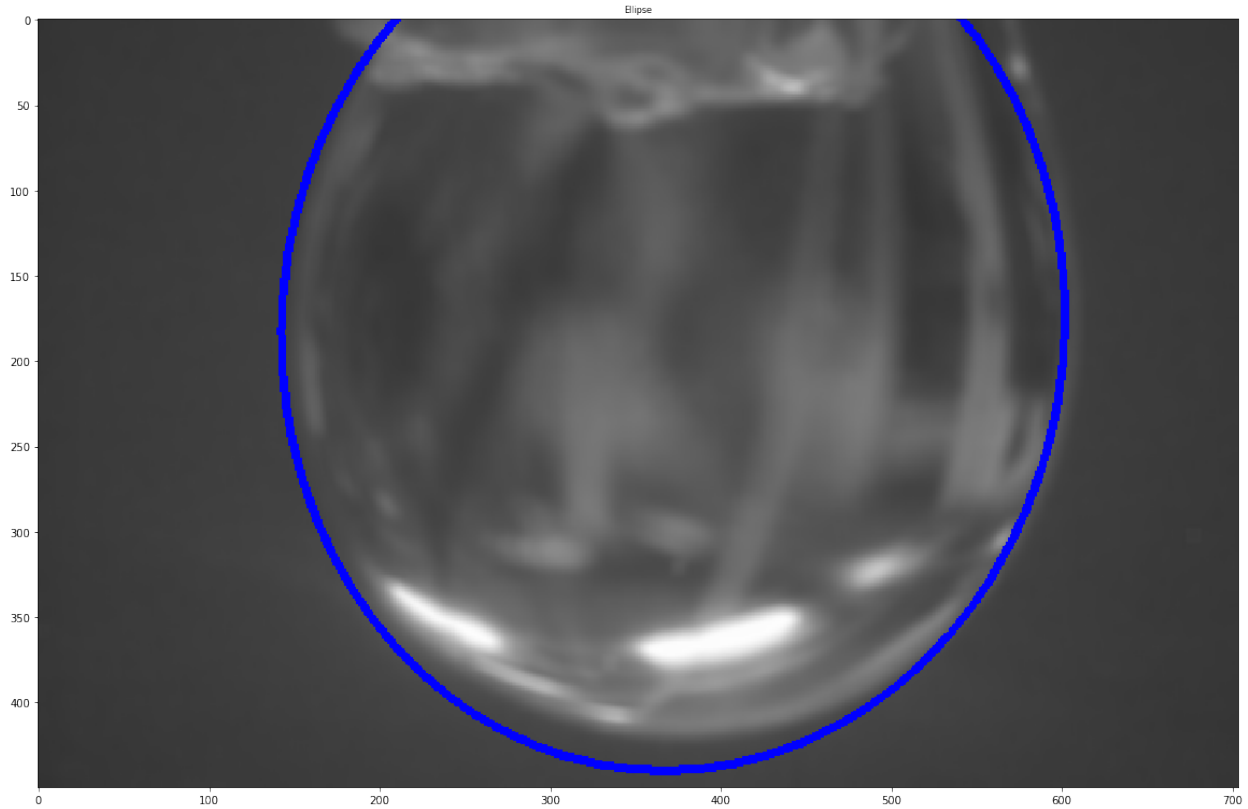


Fitting an Ellipse

```
final = np.dstack((droplet, droplet, droplet))

for cnt in contours:
    if cnt.shape[0] > 5:
        an_ellipse = cv2.fitEllipse(cnt)
        cv2.ellipse(final, an_ellipse, (0, 0, 255), 3)

plt.figure(figsize=(20, 20))
plt.title('Ellipse')
plt.imshow(final)
plt.show()
```



Credits

Citations

We kindly request that you cite the following article [\[A1\]](#) if you use project.

References

Bibliography

- [A1] Cang Zhao, Kamel Fezzaa, Ross W. Cunningham, Haidan Wen, Francesco De Carlo, Lianyi Chen, Anthony D. Rollett, and Tao Sun. Real-time monitoring of laser powder bed fusion process using high-speed x-ray imaging and diffraction. *Scientific Reports*, 7(1):3602, 2017. URL: <http://dx.doi.org/10.1038/s41598-017-03761-2>, doi:10.1038/s41598-017-03761-2.
- [B1] Cang Zhao, Kamel Fezzaa, Ross W. Cunningham, Haidan Wen, Francesco De Carlo, Lianyi Chen, Anthony D. Rollett, and Tao Sun. Real-time monitoring of laser powder bed fusion process using high-speed x-ray imaging and diffraction. *Scientific Reports*, 7(1):3602, 2017. URL: <http://dx.doi.org/10.1038/s41598-017-03761-2>, doi:10.1038/s41598-017-03761-2.

X

ximage, [27](#)
ximage.util, [11](#)
ximage.widget, [13](#)

L

`label()` (in module `ximage.util`), [12](#)
`laser_on()` (in module `ximage.util`), [12](#)
`load_raw()` (in module `ximage.util`), [12](#)

P

`particle_bed_location()` (in module `ximage.util`), [12](#)

S

`scale_to_one()` (in module `ximage.util`), [12](#)
`shutter_off()` (in module `ximage.util`), [12](#)
`slider` (class in `ximage.widget`), [13](#)
`sobel_stack()` (in module `ximage.util`), [12](#)

U

`update()` (`ximage.widget.slider` method), [13](#)

X

`ximage` (module), [13](#), [27](#)
`ximage.util` (module), [11](#)
`ximage.widget` (module), [13](#)