Normalization

January 14, 2016

In [1]:
```
# This is a cell to hide code snippets from displaying
# This must be at first cell!

from IPython.display import HTML, Image

hide_me = ''
HTML('''<script>
code_show=true;
function code_toggle() {
  if (code_show) {
    $('div.input').each(function(id) {
      el = $(this).find('.cm-variable:first');
      if (id == 0 || el.text() == 'hide_me') {
        $(this).hide();
      }
    });
    $('div.output_prompt').css('opacity', 0);
  } else {
    $('div.input').each(function(id) {
      $(this).show();
    });
    $('div.output_prompt').css('opacity', 1);
  }
  code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()"><input style="opacity:0" type="submit" value="Click here
```

Out[1]:
```
<IPython.core.display.HTML at 0x7fdea57fea50>
```

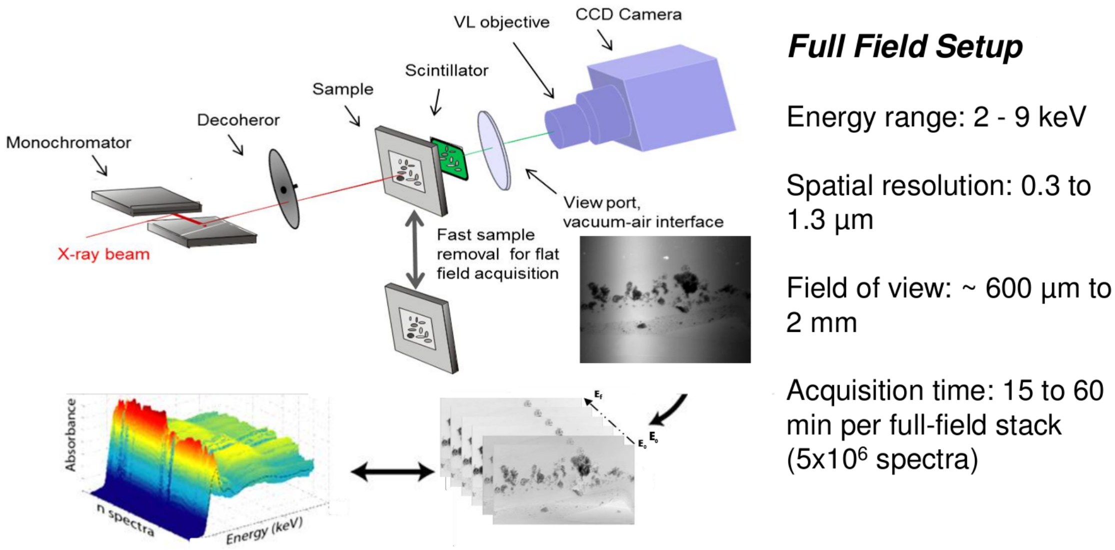
1 Image normalization

This example is part of the processing performed during a "Fullfield XANES" experiment at ESRF-ID21 and described in this article: http://iopscience.iop.org/1742-6596/425/19/192001/pdf/1742-6596_425_19_192001.pdf

During this experiement, radiographs of a sample are taken at various energies. As the full beam is used, on should divide the signal by the flat-field image, both corrected for the dark current of the CCD.

In [2]:
```
hide_me
Image(filename='files/setup.png', width=800)
```

Out[2]:



For an optimal use of the dynamic range, the data frame has a longer exposure time than the flat-field frame. Exposure time is recorded in the header of the file. Each frame needs to be corrected for the correct dark (with the same exposure time).

There are three dark-current files: dark-2.edf, dark-3 and dark-6.edf, two flat-field taken before and after the experiment and a raw data frame in the directory. Correct the image the image for the dark and flat-field effect. Check the correctness of your results often with “imshow” …

Exercise:

- Read all 6 EDF files and find their exposure time
- Correct the data and flat-field images for their respective dark-current images
- Average the two flat-field images
- Correct the data from the flat-field effect

1.1 Initialization of the scientific environment

```
In [3]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Since iPython 3 or Jupyter 4, one can also use “%pylab nbagg” to provide interactivity on the displayed images.

```
In [4]: import numpy, fabio
```

1.2 Load all images

Nota: Did you know you can call a shell command from iPython ?

```
In [5]: ls *.edf
```

```
dark-2.edf  dark-3.edf  dark-6.edf  flat-1.edf  flat-2.edf  raw.edf
```

```
In [6]: dark2 = fabio.open("dark-2.edf")
        dark3 = fabio.open("dark-3.edf")
        dark6 = fabio.open("dark-6.edf")
```

```
        flat1 = fabio.open("flat-1.edf")
        flat2 = fabio.open("flat-2.edf")
        raw = fabio.open("raw.edf")
        #print the content of the header of the "raw.edf" file
        for key,value in raw.header.items():
            print("%15s: %s"%(key, value))
```

```
HeaderID: EH:000001:000000:000000
      ByteOrder: LowByteFirst
       DataType: SignedInteger
           Size: 16777216
          Dim_1: 2048
          Dim_2: 2048
          Image: 0
   acq_frame_nb: 0
           time: Thu Dec 16 18:36:31 2010
    time_of_day: 1292520991.838942
  time_of_frame: 4.146026
         energy: 4.99
  exposure_time: 6
```

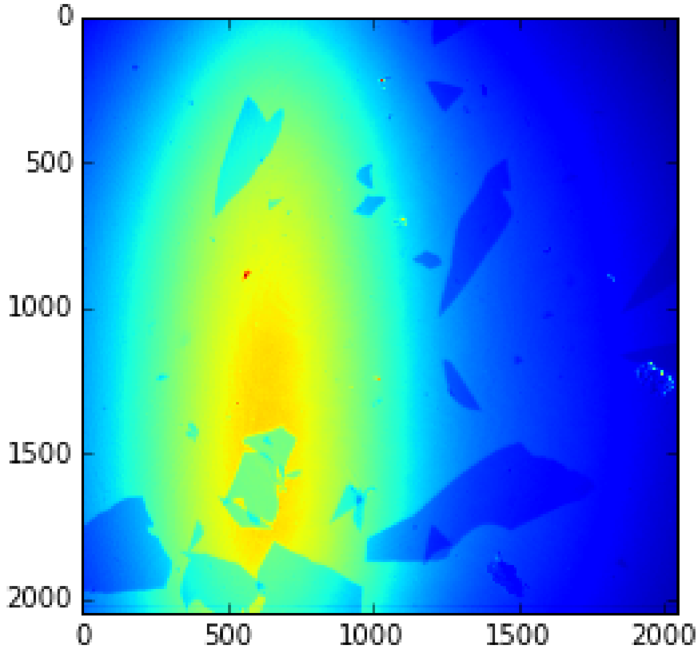
```
In [7]: #print exposure time for all images, from the metadata stored in the header
        for edf in (dark2, dark3, dark6, flat1, raw):
            print("%s : %s"%(edf.filename,edf.header["exposure_time"]))
```

```
dark-2.edf : 2
dark-3.edf : 3
dark-6.edf : 6
flat-1.edf : 3
raw.edf : 6
```

Display the image and its data-type:

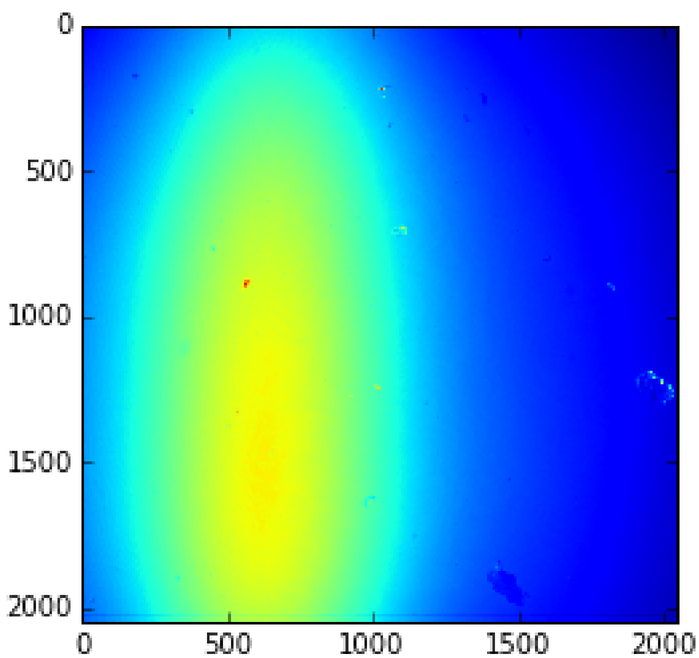
```
In [8]: imshow(raw.data)
        print(raw.data.dtype)
```

```
int32
```




```
In [9]: imshow(flat1.data)
        print(flat1.data.dtype)
```

```
int32
```



nota: check the data-type of your data as well: intergers are very bad for calculation due to over/underflow and integrer division in Python2. All scientific calculation should be performed using floating point numbers.

1.3 Dark-current correction

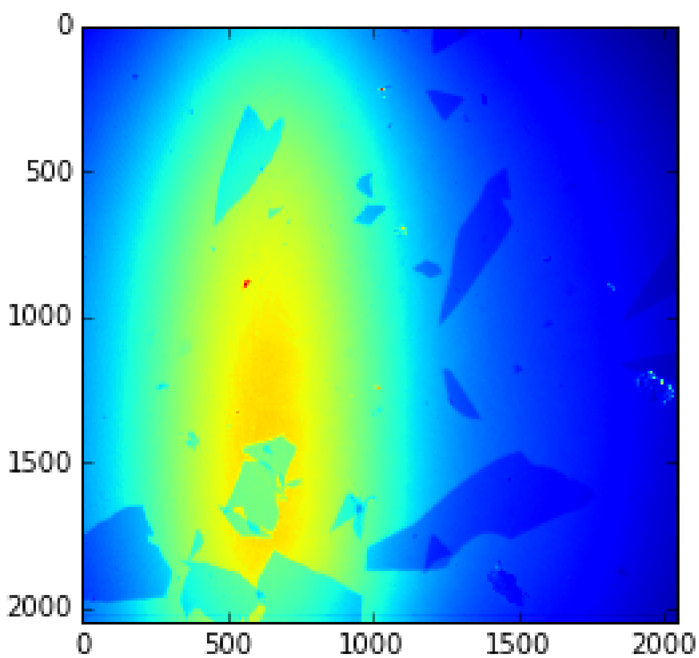
Convert the data-type for the calculation in *float*.

nota: Only one member of the equiation needs to be *float* as the other will be casted automatically to *float*

```
In [10]: raw_dc = raw.data.astype(float) - dark6.data
         flat1_dc = flat1.data.astype(float) - dark3.data
         flat2_dc = flat2.data.astype(float) - dark3.data
         print(raw_dc.dtype)
         imshow(raw_dc)
```

```
float64
```

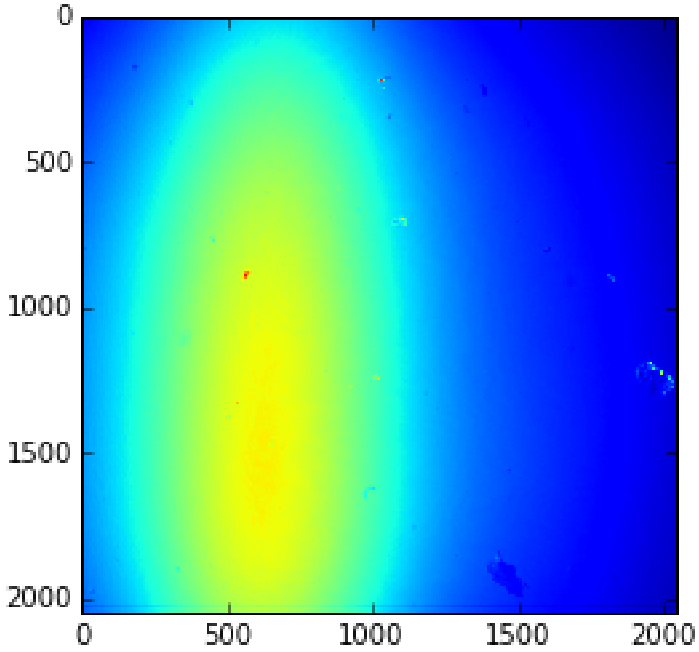
```
Out[10]: <matplotlib.image.AxesImage at 0x7fde7c579590>
```



1.4 Average out the flat-field images

```
In [11]: flat = (flat1_dc + flat2_dc) / 2.0
         imshow(flat)
```

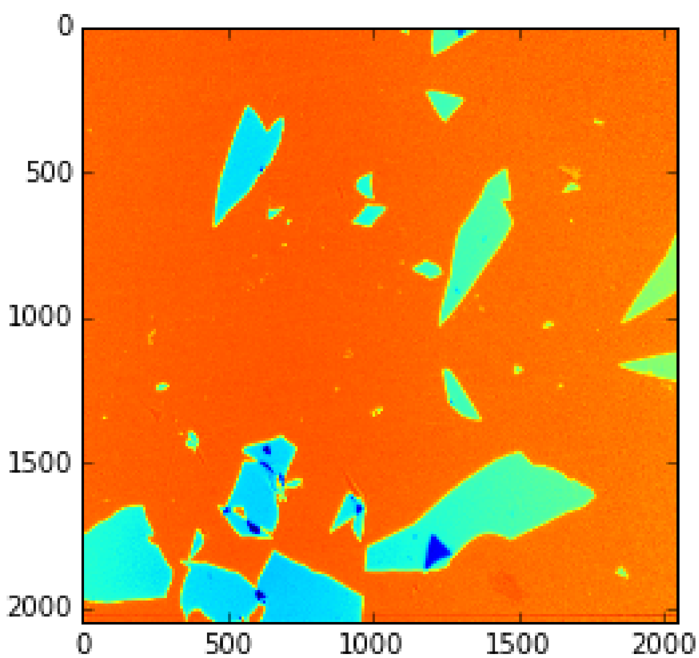
```
Out[11]: <matplotlib.image.AxesImage at 0x7fde7c4a0590>
```



1.5 Perform the flat-field correction, weighted by exposure time

In [12]:
```
#Data has been exposed for 6s, while flat was only exposed 3s
corr = (raw_dc / 6.0) / (flat / 3.0)
imshow(corr)
```

Out[12]: <matplotlib.image.AxesImage at 0x7fde7c4436d0>



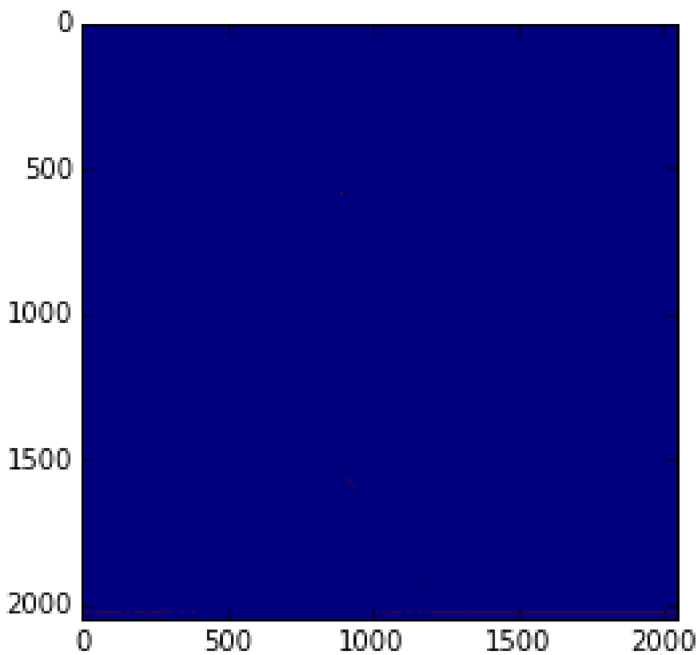
1.6 Why was it so important to calculate with floats ?

We will perform the calculation using integrers and here is the result using Python2

```
In [13]: int_corr = (raw.data - dark6.data ) / (flat1.data + flat2.data - 2*dark3.data)
```

```
In [14]: imshow(int_corr)
```

```
Out[14]: <matplotlib.image.AxesImage at 0x7fde7c366590>
```




```
In [15]: print(int_corr)
         print(int_corr.max())
```

```
[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 ...,
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]]
1
```

What happened ?

The normalisation provides values between 0 and 1 and all calculation perfomed using integers results in 0 everywhere except for 1 or a few pixels which are 1.

2 Conclusion

Most image manipulation can simply be performed using *numpy* togeather with a library to load images. This library depends on the image type. The *h5py* library is the one with the greatest future as ESRF is moving towards HDF5 and it provides a nice “dictionnary-like” interface.