

# **A high-energy surface x-ray diffraction analysis toolkit (HAT) manual**

## **1 CONTENTS**

---

2	Introduction .....	2
3	Installation/Package Versions .....	3
4	Interface Overview .....	4
5	Loading data .....	5
6	Data Processing .....	6
7	View Modes .....	8
7.1	Detector View .....	8
7.2	Transformed Detector View .....	9
7.3	Binned Projection .....	11
7.4	Export Figure .....	14
8	Scripting .....	15
8.1	Example .....	15
8.2	Function Reference List: .....	17

HAT Version 1.2

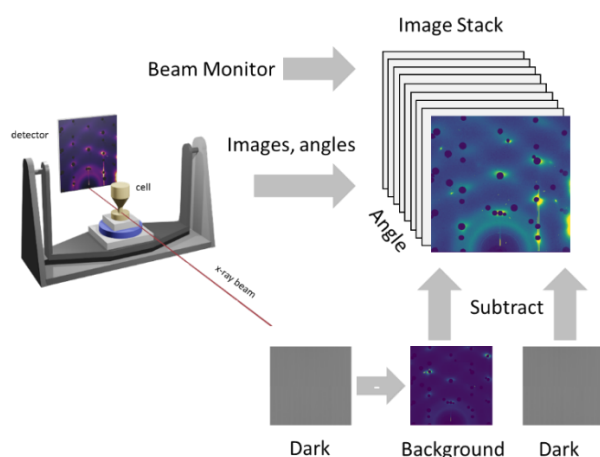
Manual Version 1.0

This is an early draft of the manual and more will hopefully be added soon. Please let me know if you have any issues.

## 2 INTRODUCTION

The High-energy surface x-ray diffraction analysis toolkit (HAT) is a cross-platform software package written in python to allow the extraction and processing of High-Energy Surface X-ray Diffraction (HESXRD) data sets. Thousands of large area detector images are collected in a single HESXRD scan, corresponding to billions of pixels and hence reciprocal space positions

Conducting a HESXRD experiment is, in principle, straightforward: one aligns a single crystal on a diffractometer so that the surface normal is perpendicular to the x-ray beam. At a grazing incidence angle (typically just above the critical angle) large area detector images are then collected while the sample is rotated about the surface normal. Since the Bragg peaks of the single crystal are many orders of magnitude brighter than the CTRs of interest we can use beam stops (such as tungsten pieces attached to magnets) to block out the intense Bragg peaks. Each image collected during the rotation of the sample is called a “frame” and corresponds to a particular sample rotation ( $\phi$ ) that is recorded by the diffractometer encoder. The whole set of images is called an “image stack”. Depending on the detector type we may wish to subtract a dark image from each frame. Furthermore, a background image, such as the sample environment without the sample, can also be subtracted and this may also have its own dark image. Each image collected should also be normalised to a beam intensity monitor such as: an ion chamber or the synchrotron ring current. This Image Stack representation is represented schematically in Figure 1.



**Figure 1. Schematic representation of a typical data set from a HESXRD experiment.** A stack of images is collected in the experiment, this can be operated through subtraction of dark images and backgrounds as well as rotation and flipping. Each image should correspond to a different sample angle.

### 3 INSTALLATION/PACKAGE VERSIONS

---

You will need a python environment and an editor

For windows I recommend (Visual Studio Code).

The version of python currently tested is 3.8.5.

Below is a table of packages needed and what version is known to work with all other packages listed.

Package Name	Version most recently tested (Other versions will likely work)
Numpy	1.19.3
Scipy	1.5.4
PyQt5	5.15.4
Pyqtgraph	0.11.0
Matplotlib	3.3.3
Fabio	0.10.2
Importlib	1.0.4
Numba	0.51.2
H5py	3.1.0
hdf5plugin	2.3.1

You can install these from a terminal (in visual studio code go to terminal -> new terminal) using pip.

e.g.

`pip3 install numba, numpy, scipy, PyQt5, pyqtgraph, matplotlib, fabio, importlib, h5py, hdf5plugin`

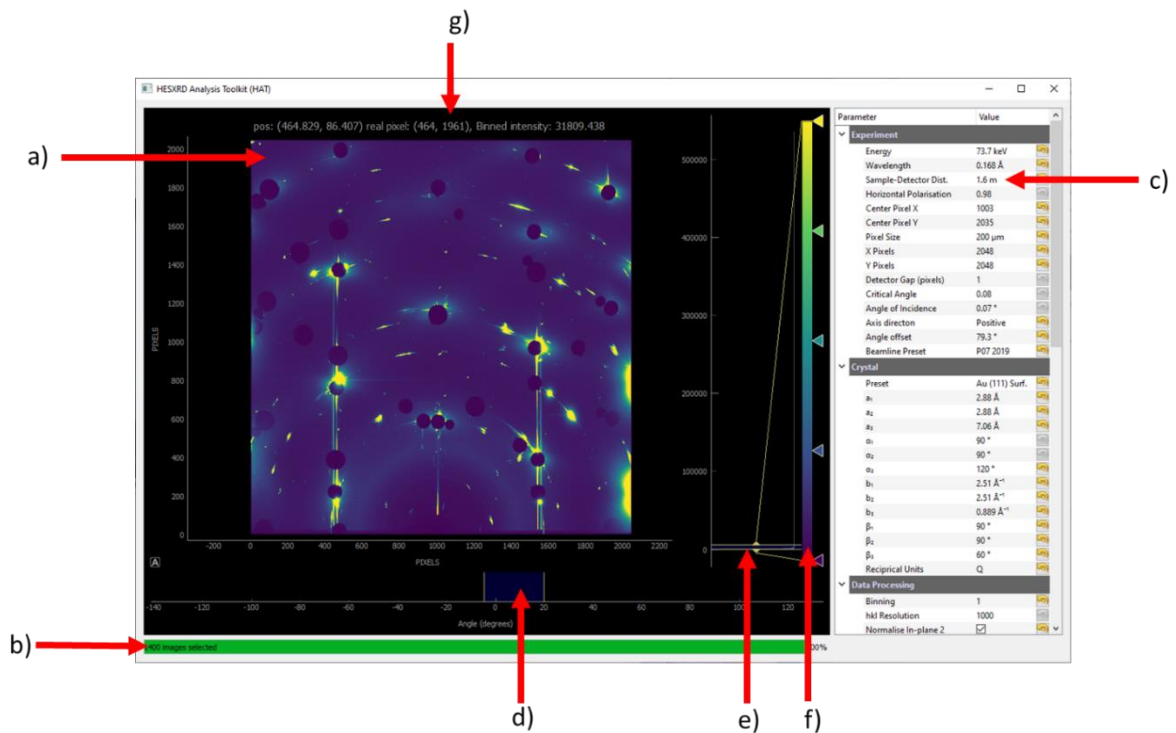
If your computer has a graphics card that supports CUDA you should also install the nvidia sdk:

<https://developer.nvidia.com/cuda-downloads>

With the path then inside the HAT directory you should then run main.py. i.e.

`$ python3 main.py`

## 4 INTERFACE OVERVIEW



a) Image stack view, either maximum intensity for each pixel or summed intensity.

b) Progress bar and notification area.

c) Parameter Tree and command buttons.

d) Angle/frame range.

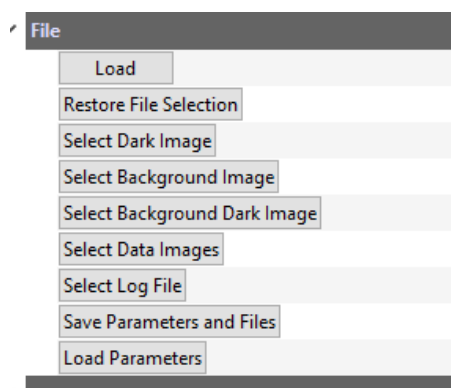
e) Intensity range selector.

f) Color map customization.

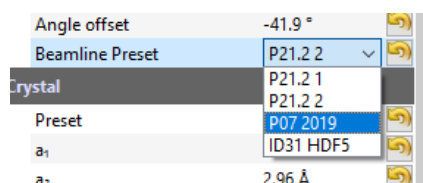
g) pixel hover information.

## 5 LOADING DATA

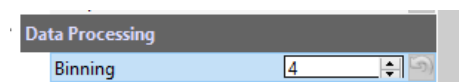
---



Images can be selected using the File command buttons. Different options are available depending on the beamline present chosen in the Experiment parameters.



All images will be binned according to the “Binning” parameter under Data Processing. If you have many images, you may not be able to load them all into system memory if the Binning number is too low.



**Load** – This loads all the files selected into memory, do this after selecting all the required files

**Restore File Selection** – If you have previously pressed “Save Parameters and Files” this will restore the previously selected files. Note: You still need to press “Load”.

**Select Dark Image** – Allows you to select a dark image, this is then subtracted from each image that is loaded. Note: The dark image for the P07 beamline preset is read in automatically.

**Select Background Image** – Select an additional image to subtract from each image

**Select Background Dark Image** – Select a dark image for your background image

**Select Data Images**– Select a number of images to loaded into memory

















**Select Log File** - This is very dependent on the beamline preset, for P07 it will be read in automatically. For Manual mode you should provide a 3-column csv file. Col 1= filename, Col 2, Angle, Col 3 = Monitor.

**Save Parameters and Files** – This will save the current parameter tree (to params.bin) and file selection (to imagestack.bin).

**Load Parameters** – If you have previously pressed “Save Parameters and Files” this will restore the previous parameters (but not the files). This reads from the params.bin file.

## 6 DATA PROCESSING

---

Data Processing		
Binning	4	
hkl Resolution	1000	
Divide Bins By Frequency	<input checked="" type="checkbox"/>	
In-plane Grid Size	800	
White background	<input type="checkbox"/>	
Mean Images instead of Max	<input type="checkbox"/>	
Acceleration	cuda (gpu)	
Bin From Full Images	<input type="checkbox"/>	
Apply Intensity Corrections	<input type="checkbox"/>	
Correct for Refraction	<input type="checkbox"/>	
Intensity offset	0	
Multiply intensity by	1	
Image Rotation	180 degrees	
Image Flip U/D	False	
Image Flip L/R	False	
Use 2nd detector	<input type="checkbox"/>	
<input type="button" value="Run Script"/>		

This section allows the user to select various options on how the data is read in and outputted.

**Binning** - Binning size when data is read into memory

**Hkl Resolution** – Resolution of image stack view (a) after coordinate transformation using the Transformed Detector Image view, this is usually fine at 1000.

**Divide Bins By Frequency** – You should normally do this, essentially it normalizes any bins (Binned Projection) by the number of times a pixel's intensity is placed in that bin. This makes sense since some bins will have several pixels falling into them whereas others will have few.

**Grid Size** – This is the number of bins in each direction, so a value of 1000 would give a projection consisting of 1000x1000 bins.

**White Background** – Toggle to change the background of the application window to white.

**Mean Images instead of Max** – The standard way is to show the maximum intensity for each pixel across the whole stack. However, an alternative is to sum all the images (we take the mean, so the intensity doesn't massively increase). Normally the background becomes very high, and you have to carefully select a small angular range to be able to see CTRs.

**Acceleration** – Some operations can be quite slow. It is possible to speed these up using the numba library which compiles the code at run time, but of course you need to have access to this package.

You can also select to use CUDA, which then runs some code on the GPU. This requires a compatible GPU and the correct libraries available. This increases the speed of the binning operations. However, the Transformed Detector Image view currently uses scipy's interpolate function, which is CPU based, so this is not as affected by using the GPU. However binned projections should generate much quicker.

Bin From Full Images – Typically the images need to be binned (see Binning) to be able to fit into memory, this effectively reduces the number of pixels that can then be binned into different projections. Often this is barely noticeable, but if it is an issue you can tick this box and then each image will be reloaded (without binning) while generating the projection. Only one unbinned image will be stored in memory at once. Obviously if Binning=1 it would make no sense to do this as you would get the same result (after a longer wait).

Apply Intensity Corrections – The intensity of either the Transformed Detector Image or a reciprocal space projection will be corrected for the following factors if this is checked.

Polarization, Lorentz Factor, Change in distance due to flat detector and curved Ewald sphere, Beam inclination and Rod interception.

Correct for refraction – This will correct the qz component of momentum transfer for refraction using the “critical angle” parameter under experiment. This could be useful for thin-films.

Intensity offset – This adds a constant value to the intensities of images loaded which might be useful for avoiding negative values. Although it is also currently applied to all images including darks and background.

Multiply Intensity by – Multiplies the intensity by the value, should be useful if for example you have a fitting program that struggles with small numbers.




Image rotation – You can rotate the detector images by multiples of 90 degrees.

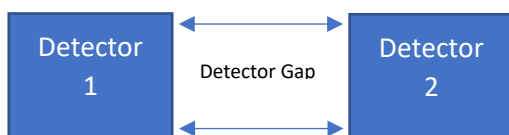
Image Flip U/D – Flip the detector images up/down. Applied after image rotation.

Image Flip L/R – Flip the detector images left/right. Applied after image rotation and Flip U/D.

Use 2<sup>nd</sup> detector – HAT currently supports two detectors side by side (but not for HDF5 files).

Every time you select an image file you will be asked again for the 2<sup>nd</sup> detector files. In the end this will be stitched together to behave as one image. You should also set the gap between the two detectors in the Experiment parameters (in terms of pixels). Both detectors should be the same size and orientation. If you change the Detector Gap you need to press “Load” and reload all the files.

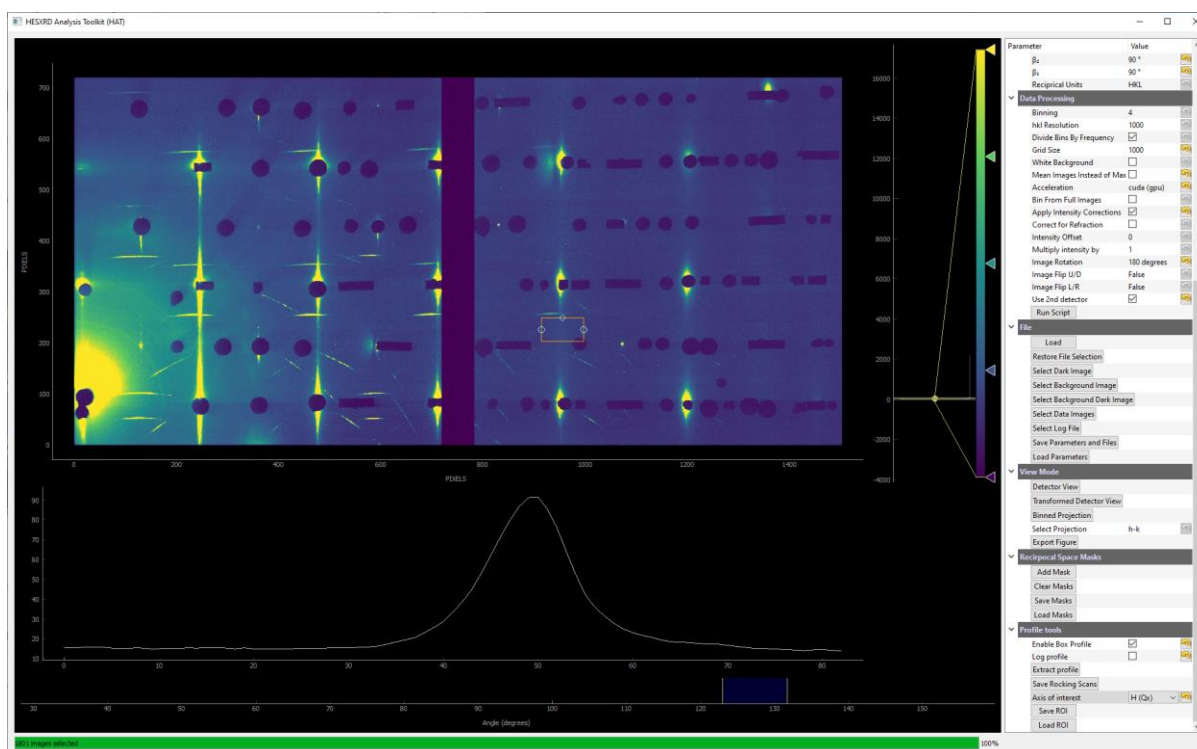
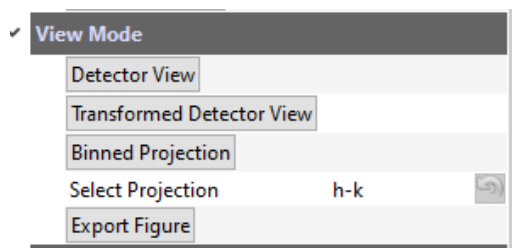
Y Pixels	2880	
Detector Gap (pixels)	1	
Critical Angle	0.08	



Run Script – This will reload the file script.py and run the function script\_main(). See the section on scripting.

## 7 VIEW MODES

### 7.1 DETECTOR VIEW



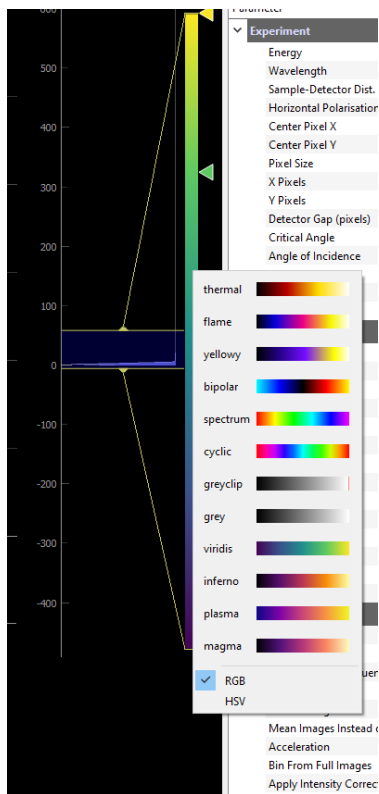
This view shows the detector images (after binning, dark, background subtraction, etc.). The intensity is shown either as the maximum intensity across the angular range or the average intensity across the angular range (see [Mean Images instead of Max](#)).

This is the only mode that one can change the angular range which determines which images are included in any binning.

It is also the mode that rocking scans can be extracted, simply select the angular range (this is the x-axis on a rocking scan), then draw a ROI. The intensity will be averaged along the x-direction of your region of interest (so don't make it too wide). Then for each pixel in the vertical direction a rocking scan will be saved.

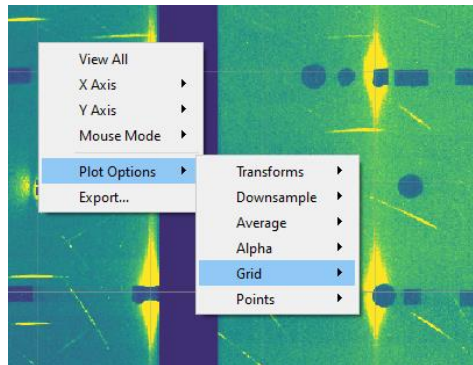
It is also possible to extract line profiles, hovering over a pixel shows the pixel number and intensity.



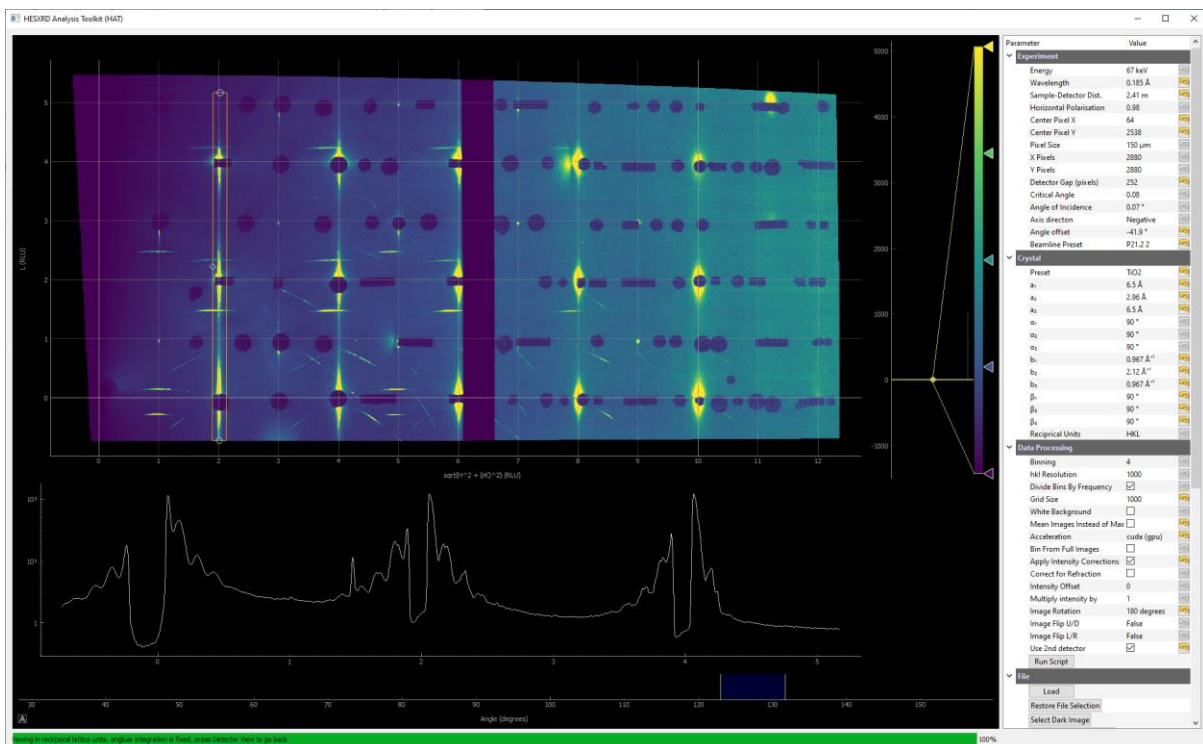


As with all view modes you change the colormap by right clicking on the color gradient and selecting a new colormap. You can also change the color scale range (color min/max) by dragging the two sliders on the histogram.

If you right click on the image there are various options for changing the grid etc. You can also do a basic export, although you will get better results using the custom “Export Figure” button in the parameter tree.



## 7.2 TRANSFORMED DETECTOR VIEW

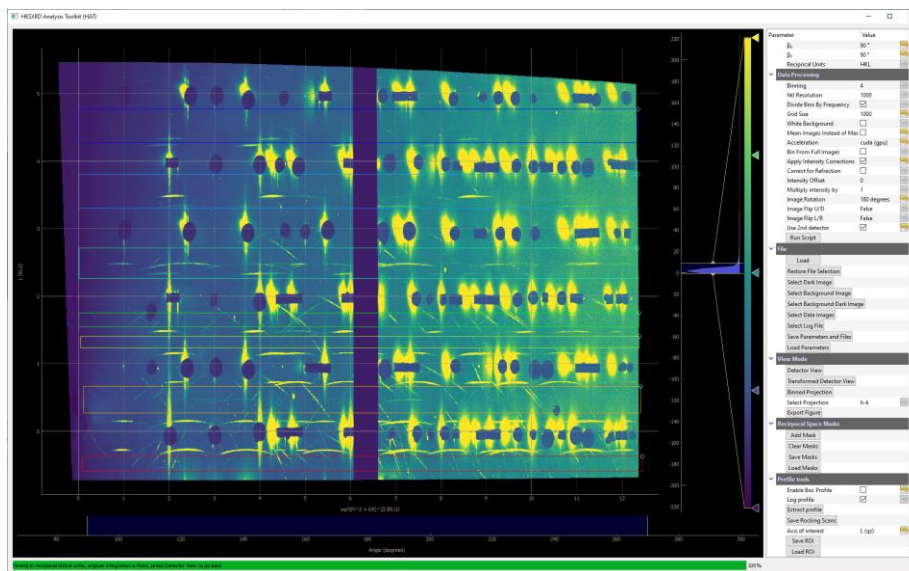


In some cases, it's useful to transform a detector image (or sum/max of multiple images) into a useful coordinate system. This can produce nice figures and allow extraction of some data with an attached coordinate system.

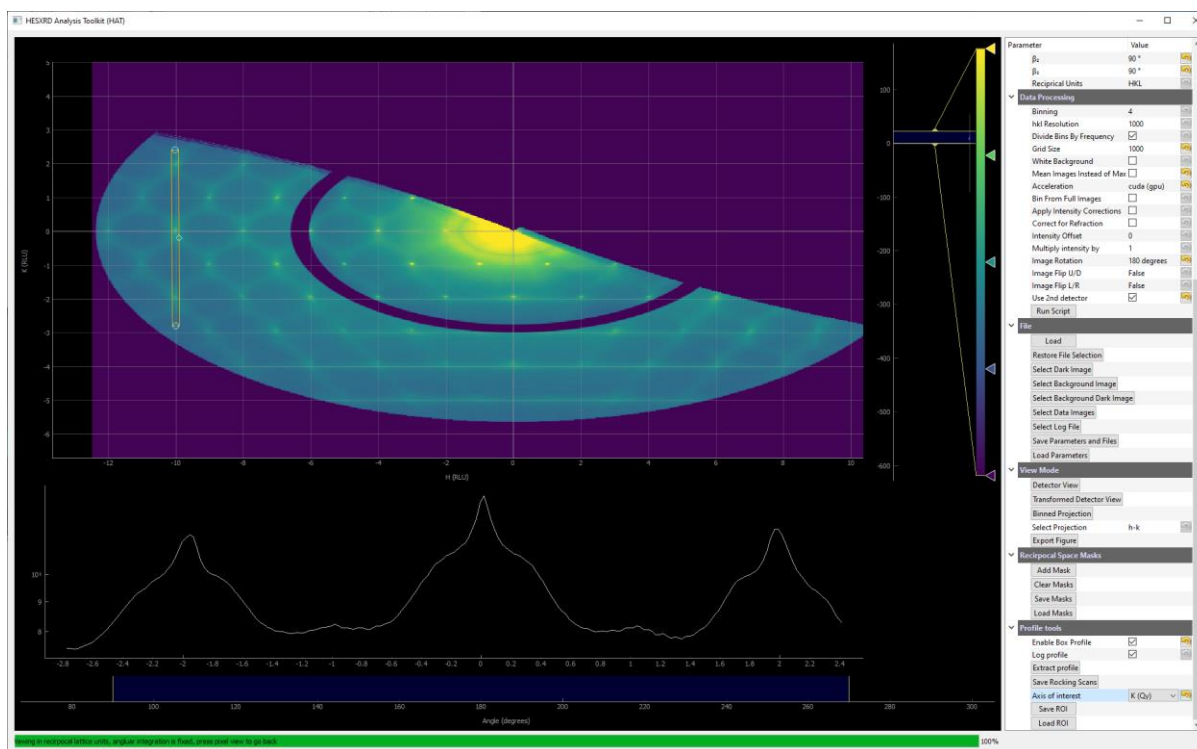
For cubic systems (in surface units) the x-axis becomes  $\sqrt{h^2 + k^2}$  and the y axis L. If the lattice parameter  $a_1$  does not equal  $a_2$  but all the angles remain 90 degrees, then the x-axis will be  $\sqrt{h^2 + (rk)^2}$  where  $r = a_1/a_2$ . For hexagonal systems where  $\alpha_1 = \alpha_2 = 90$  and  $\alpha_3 = 120$ , the x-axis is  $\sqrt{(h\phi_1/b_1)^2 + (k\phi_1/b_2)^2}$  – essentially the transformation matrix is not applied and the  $q_x$  and  $q_y$  vectors are instead normalized by  $b_1$  and  $b_2$ . Other systems are currently not implemented, but you can always change the reciprocal units to “Q” instead.

The transformed detector view is also helpful for applying masks when generating a H-K projection (i.e. in-plane map). You can add as many masks as you need, then any pixels which fall within them will be included in the binning, pixels outside them will not be used.

(Remember to “Clear Masks” if you change projection mode).



## 7.3 BINNED PROJECTION



Several different projection modes can be chosen. The first one is h-k, this mode generates a reciprocal space map using any pixels that fall within the qr and qz values of any masks applied. For none cubic systems this might become extremely distorted as HAT does not currently support axis that have different angles than 90 degrees between them.

In such a situation it is better to great the map in Q units and then manually handle the transformation when creating a figure. As shown in this code snippet:

```
#inplane map
def plot_projection_hk(hat, grid_h, grid_k, grid_i, cmin, cmax, outfile_name):
    #the easiest way to do none cubic/rectangular system to work in
    #Q units and then convert when plotting.
    def tr(h, k):
        h, k = np.asarray(h), np.asarray(k)
        return (np.sqrt(3)/2)*h*2.51, (k+0.5*h)*2.51

    def inv_tr(x, y):
        x, y = np.asarray(x), np.asarray(y)
        return (x/(np.sqrt(3)/2))/2.51, (y-0.5*x)/2.51

    plt.rcParams.update({'font.size': 8})
    plt.rc('legend', fontsize=8, handlelength=2)
    plt.rcParams.update({'font.sans-serif': 'Arial'})
    cm = 1/2.54 # centimeters in inches
    fig = plt.figure(figsize=(20*cm, 20*cm), dpi=600)
```

```

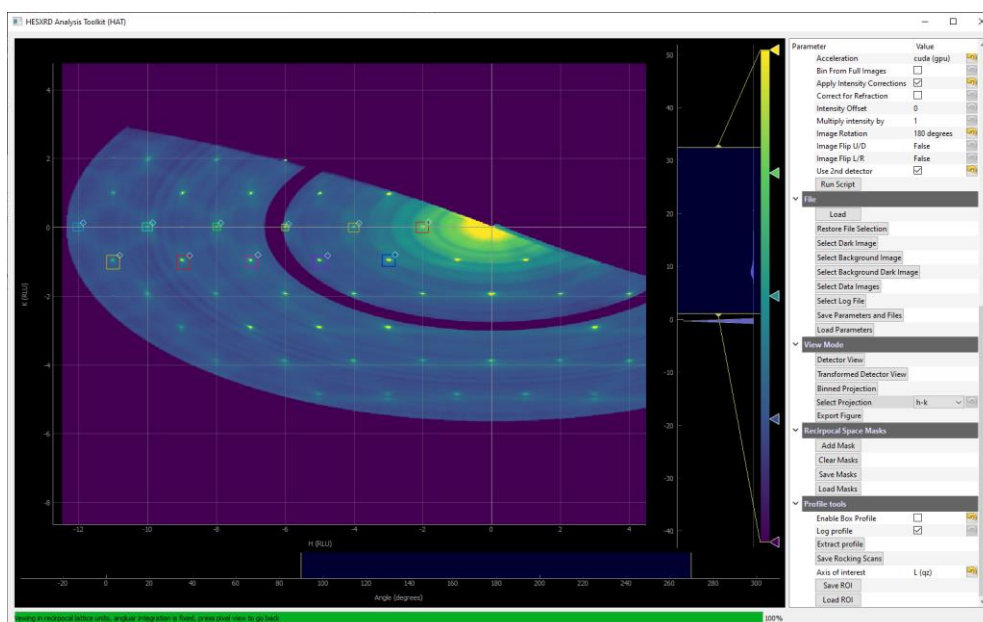
grid_helper = GridHelperCurveLinear((tr, inv_tr), grid_locator1=FixedLocator([0.9, 0.95, 1, 1.05, 1.1]), grid_locator2=FixedLocator([-0.1, -0.05, 0, 0.05, 0.1, 1]))
ax1 = Subplot(fig, 1, 1, 1, grid_helper=grid_helper)
fig.add_subplot(ax1)
grid_i = grid_i.T.astype(float)
qmax = np.max(hat.grid_qr)
ax1.imshow(grid_i, extent=(-1*qmax, qmax, -1*qmax, qmax), origin='lower', vmin=cmin, vmax=cmax, cmap='viridis', interpolation='bicubic')

plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=0.6, hspace=0.8)
plt.xlabel('$H$ (RLU)')
plt.ylabel('$K$ (RLU)')

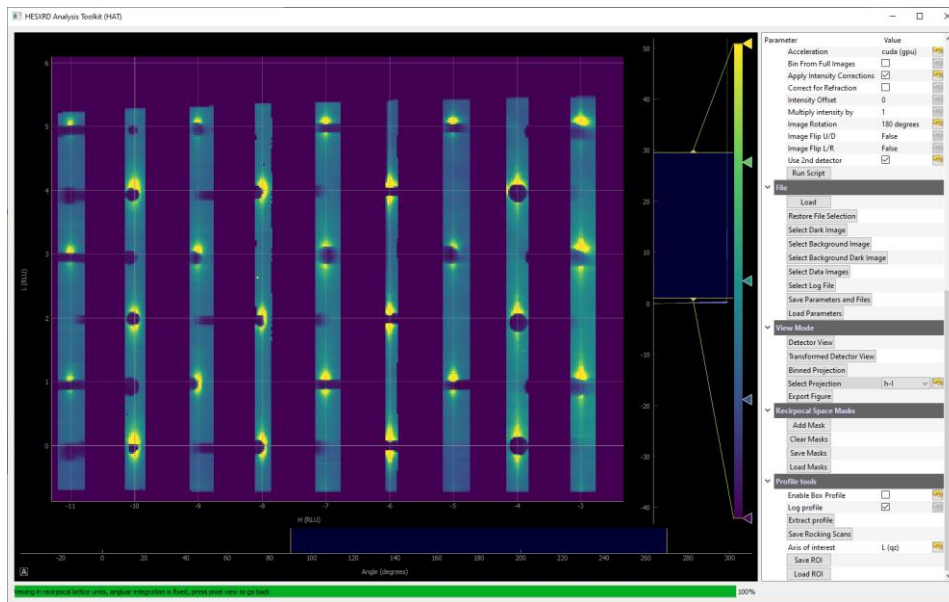
```

Essentially the plotting uses matplotlib's "grid\_helper", to generate slanted axes for hexagonal coordinates (i.e. 111). One has to define a transformation and inverse transformation function. The data is in momentum-transfer units (Q) but is then scaled through dividing it by the reciprocal lattice vector,  $2.51 = b_1 = b_2$ .

One can also use the h-k projection for placing masks to generate the other projections. This is useful when trying to extract CTRs. (Remember to press Clear Masks, otherwise you will still have to mask you used in the "Transformed Detector View" active)



This image shows an extreme example. If we wanted to extract CTRs at each position with a mask box around it, we would have to recalculate the binned projection 11 times which would be very time consuming. A more efficient workflow is to project the masks in the h-l plane.



The result is then as above. Each separate part of the image corresponds to a separate CTR (although not so much in this case as the signal is very weak for most of them). Since the data has already been binned and the intensities appropriately corrected, the CTRs can then just be extracted one at a time using the box profile tool (one should also take nearby background regions for each CTR and subtract those from the profiles).



Of course, if we had instead chosen a  $k-l$  projection most of the CTRs would have been on top of each other as they either are on the line  $k=0$  or  $k=-1$ . This is why there are some many different projection modes.

## 7.4 EXPORT FIGURE

Export figure allows the data in the current view to be plotted and manipulated. Essentially you chose a file name and the file plotting.py is reloaded, then depending on what your current view a different function from plotting.py will be called. This function is passed the coordinate grids, the data, the minimum and maximum color values, and the filename that has been given.

It's then pretty much up to the user what they want to do with that, but the idea is (and the examples do) to use matplotlib to make some publication quality figures. It would however be possible to perform some additional data analysis here, you have the data and you have python ..

This is an example plotting function for the transformed detector view.

```
def plot_transformed_detector(grid_hk,grid_l,grid_i,cmin,cmax,outfile_name):
    plt.rcParams.update({'font.size': 8})
    plt.rc('legend', fontsize=8, handlelength=2)
    #see below for how to install fonts
    plt.rcParams.update({'font.sans-serif': 'Arial'})
    cm = 1/2.54 # centimeters in inches
    fig = plt.figure(figsize=(20*cm,20*cm),dpi=600)
    ax1 = Subplot(fig, 1, 1, 1)
    fig.add_subplot(ax1)

    ax1.imshow(grid_i.T, extent=(grid_hk.min(),grid_hk.max(),grid_l.min(),grid_l.max()), origin='lower',
    vmin=cmin,vmax=cmax,cmap='viridis')
    #ax1.grid(True,linewidth=0.05,color='w')
    ax1.set_aspect(1)
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=0.6, hspace=0.6)
    ax1.xaxis.set_major_locator(ticker.MultipleLocator(1))
    plt.xlabel('$\sqrt{H^2+{(rK)}^2}$ (RLU)')
    plt.ylabel('$L$ (RLU)')
    plt.savefig(outfile_name)
    plt.close()
```

The plotting.py file will be reloaded every time you press the “Export Figure” button so if you make a mistake you just need to change the plotting file.



## 8 SCRIPTING

---

HAT allows for rudimentary scripting, in that the “Run Script” button allows a python file to be loaded and a function (script\_main()) to be executed). In this way one has direct access to the HAT software and can call its various functions in sequence. You can also call any other python functions you wish as well.

### 8.1 EXAMPLE

An example script.py file:

```
import glob
import numpy as np
import math

def script_main(hat):
    hat.restore()
    folder_list = glob.glob("D:/CV_NaOH_2/*")
    for i, folder in enumerate(folder_list):

        #get list of file names in directory ending with tif
        file_list = glob.glob(folder+"/*.tif")

        #filter out any dark images
        files_no_dark = [x for x in file_list if "dark" not in x]

        #load files
        hat.image_stack.select_images(files_no_dark)
        hat.load()
        hat.setUnits(0) #q units

        #make hkl map
        hat.detectorTransform()
        hat.setColorMap("viridis")
        hat.loadMasks("quickrotate.msk")
        hat.setProjection(0)
        hat.makeProjection()
        hat.setLevels(20, 70)
        hat.makeFigure("D:/output/"+str(i))
```

Here the entry point is script\_main(hat). Hat is the HAT window class and allows any of the interface functions to be called. There are several helper functions for scripting also. Here we will go through what this script is doing:

```
import glob
import numpy as np
import math
```

We load some python libraries we would like to use in our script, glob lest you use wildcard like syntax for directories.

```
def script_main(hat):
    hat.restore()
```

We restore the parameter tree we have saved beforehand; this is the same as pressing the “Load Parameters” button on the GUI.

```
folder_list = glob.glob("D:/CV_NaOH_2/*")
for i,folder in enumerate(folder_list):

    #get list of file names in directory ending with tif
    file_list = glob.glob(folder+"/*.tif")

    #filter out any dark images
    files_no_dark = [x for x in file_list if "dark" not in x]
```

The path D:\CV\_NaOH\_2 contains several directories, each of which is a directory containing a measurement (i.e. images collected during a rotation).

We use glob to great a list of all the folder names, and then we loop through each folder.

For each folder we put all the files ending in \*.tif into a list, we then create a list containing those file names that don't contain the string “dark”, i.e., we are filtering out the dark images.

```
#load files
hat.image_stack.select_images(files_no_dark)
hat.load()
hat.setUnits(0) #q units
```

Then we load the images in that folder by passing the list of images to image\_stack, and then calling the load function. This is equivalent to selecting several files with the “Select Data Images” button and then pressing “Load”.

We also set the reciprocal space units we want to use to Q.

```
#make hkl map
hat.loadMasks("quickrotate.msk")
hat.setProjection(0)
hat.makeProjection()
hat.setColorMap("viridis")
hat.setLevels(20, 70)
hat.makeFigure("D:/output/"+str(i))
```

Next we transform the load a previously setup mask file and calculate an in-plane projection. We choose a colourmap, set the colour min/max values and then make a figure, which is defined in plotting.py.

For each folder we will then generate a numbered image in D:\output showing the in-plane map.



## 8.2 FUNCTION REFERENCE LIST:

Below is a list of some useful functions to use in scripts.

Function Name	Description
<b>views</b>	
hat.makeProjection()	Create binned projection
hat.detectorTransform()	Apply coordinate transform to detector View
hat.detectorMode()	just show the max/sum of the images using pixel coordinates, i.e. Detector View
hat.setProjection()	set the projection mode 0: hk, 1: h-l, 2: h-k, 3: $\sqrt{h^2+k^2}$ -l
hat.makeFigure(filename)	save a figure using plotting.py, filename will be the output file name
hat.setLevels(cmin, cmax)	set the histogram minimum and maximum colour levels
hat.setAngleRange(start, end)	set the angle region, can only do in pixel view
hat.setColorMap(colormapname)	viridis is one example color map you can set others (not so useful as you have to set your own colormap in plotting.py)
hat.setUnits(value)	value is either 0 for Q or 1 for HKL
<b>masks</b>	
hat.clearMasks()	Clear and currently loaded masks
hat.loadMasks(filename)	load masks from a file called filename
<b>files</b>	
hat.restore()	load saved parameters
hat.restoreImageStack()	reselect last saved files
hat.image_stack.select_images(files, files2)	select the images to load, files is a list of image names (with path) to load files2 is an optional 2nd list if there is a 2nd detector if hdf5 file files should be a string and not a list
hat.image_stack.select_dark(file1, files2)	similar syntax to above, dark is subtracted from each image
hat.image_stack.select_background(filename, filename2)	similar syntax to above, background is also subtracted from each image
hat.image_stack.select_background_dark(filename, filename2)	similar syntax to above, dark is subtracted background image
hat.image_stack.read_log_file(filename)	read log file, only works for some predefined beamlines
hat.load()	load the files defined in image_stack
<b>profile tools</b>	
hat.setProfileLog(value)	Pass either true or false to change if the profile should have a log y axis

hat.enableProfile(value)	Pass either true or false to enable/disable the box profile
hat.loadroi(filename)	load a ROI from a file called filename
hat.extractrois(filename)	save roi as xy ascii to filename
hat.saverocks(foldername)	save rocking scans form region of interest, pass a folder name each file is a csv with angle and intensity for each vertical line along y
hat.setProfileAxis(value)	Change the axis of interest for the profile H (Qx)" = 1, "K (Qy)" = 2, "HK (Qr)" = 3,"L (qz)" = 4 (Default is 4)