Contents

1 OptiJ Manual

This Manual contains:

- The OptiJ software requirements
- The OptiJ installation guide
- A manual and description of the 8 Plugins in the OptiJ Library.

1.1 Software Requirements

All the plugins in OptiJ require ImageJ, Java SE SDK and lwjgl to be installed. These can be installed as follows.

1. Download ImageJ with Java v1.8 or greater. It is recommended to install this from: https://imagej.nih.gov/ij/download.html bundled with Java.
2. The Java SE Development Kit 8 can be downloaded from: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html if required.
3. Check that ImageJ is using the correct version of Java by running the application and hovering over the status bar as shown in figure 1.

4. If ImageJ is using the wrong version of Java please follow the guide at: http://imagej.net/Troubleshooting.

5. The GPU acceleration feature requires a number of native libraries to be utilised. These can be installed from: https://sourceforge.net/projects/java-game-lib/files/Official Releases/LWJGL 2.9.3/ (the folder downloaded should read: lwjgl-2.9.3.zip). The required native files are as shown in figure 2.

6. To ensure that ImageJ can locate these native libraries install the files relevant to your operating system in your ImageJ library folder (figure 3). This can be done by dragging the files from the relevant folder to your ImageJ Library. Figure 4 shows an example of this using Mac OS where the files under lwjgl-2.93 -> native -> macosx are copied to imagej -> jre -> lib. For Windows replace macosx with Windows.

7. Windows users might also need to add this ImageJ library folder to their system environment variables path. To do this locate *system variables* in *environment variables* and click on the 'Path' parameter (figure 5). Press edit and add a new path pointing to your ImageJ library folder, example: C:/Documents/ImageJ/jre/lib.

8. This completes all the software requirements. The OptiJ plugins can now be installed and run.

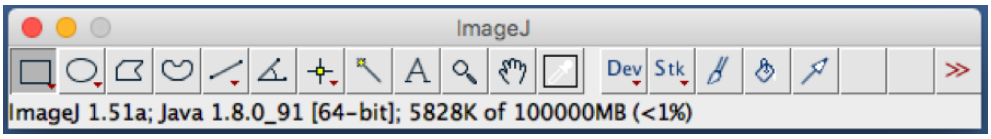

Figure 1: Checking that ImageJ is using the correct version of Java - 1.8.0_91 is this case.

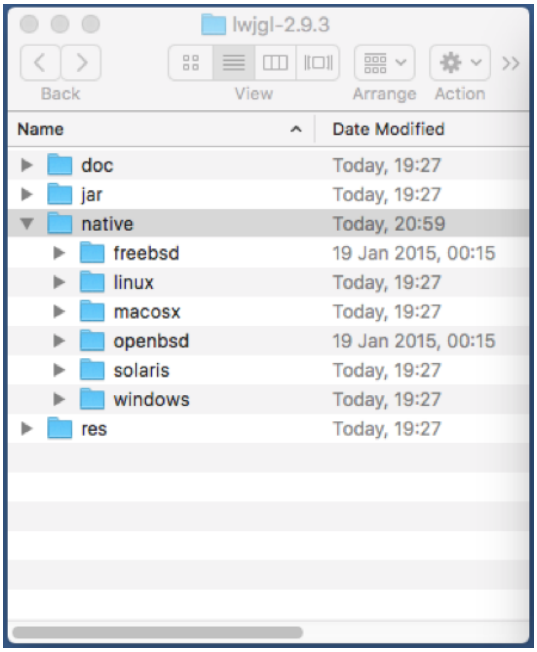

Figure 2: The native files required to enable GPU/CPU acceleration in the 2D Reconstruction plugin.

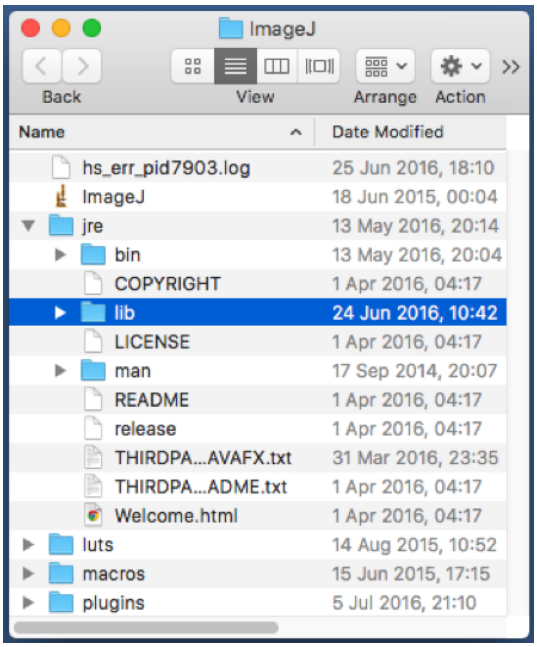

Figure 3: ImageJ library folder.

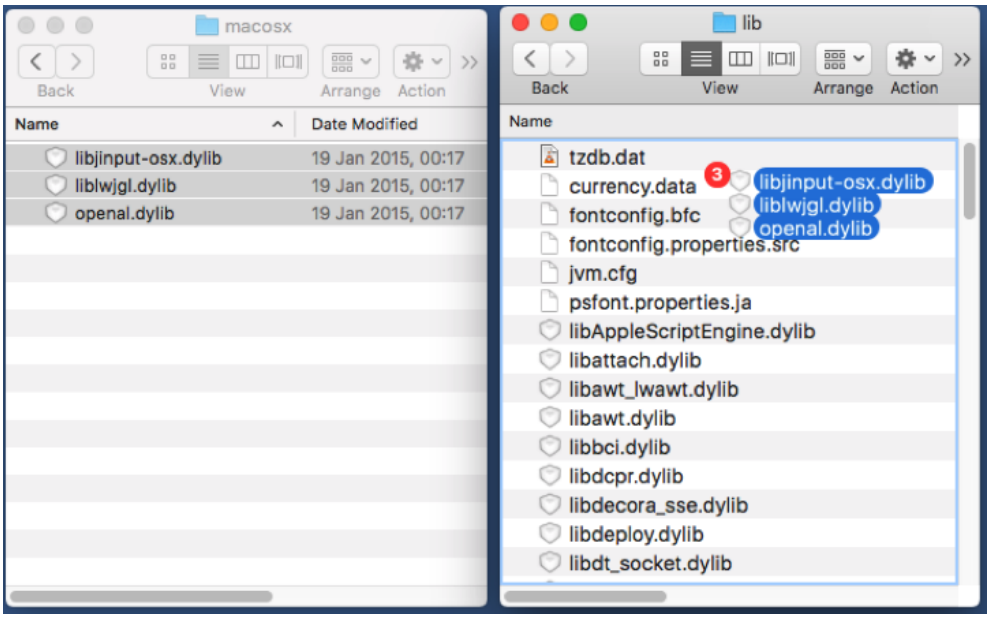

Figure 4: Copying the downloaded native library files to the ImageJ library folder (use the files relevant to your OS).

1.2 OptiJ Plugin Library Installation

All the plugins are bundled into a single *.jar* file called *OptiJ.jar*. To install this:

1. Download the OptiJ ImageJ plugin library *OptiJ.jar* at: <to be determined by Laser Analytics Group>.com.
2. Drag and drop the downloaded jar file into ImageJ as shown in figure 6.
3. Save the .jar file in the plugins folder as shown in figure 7.
4. Close ImageJ and reopen it.
5. The OptiJ plugins should now appear under the *Plugins* tab in ImageJ as shown in figure 7. There should be 8 plugins: Derive Focus Measure; Estimate Background; Beer-Lambert Correction; Estimate Tilt & Static Offset;

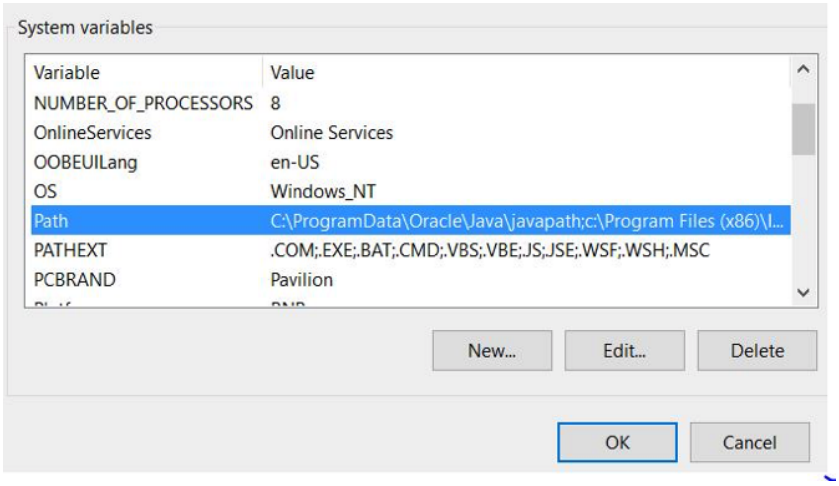

Figure 5: *Windows System variables*. This is part of *environment variables* in Windows.

Image Noise Estimation; Dynamic Offset Correction; Create Sinogram; 2D Reconstruction.

6. This completes the OptiJ plugin installation. The plugins can now be utilised.

The plugin manuals are presented in the following order:

1. Derive focus Measure
2. Create Sinogram
3. 2D Reconstruction
4. Estimate Background
5. Beer – Lambert Correction Plugin
6. Estimate Tilt & Static Offset
7. Image Noise Estimation Plugin
8. Dynamic Offset Correction Plugin

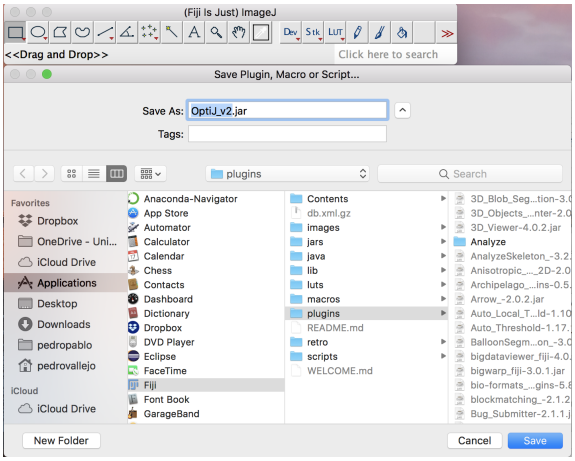

Figure 6: Installing the OptiJ Plugin Library: simply drag and drop the *OptiJ.jar* file in ImageJ and save it in the *plugins* folder.

1.3 Derive Focus Measure Plugin

The Derive Focus Measure Plugin uses the Tenengrad variance to calculate the focus measure in each input image. The GUI and output logs are as shown in figures 8 and 9 respectively.

1.3.1 Using the Plugin

1. Drag and drop the required image or image stack into ImageJ.
2. Press ok to start.

1.3.2 Processing Sequence and Example

1. The plugin first displays the employed focus measure algorithm and reference respectively.
2. The Tenengrad variance is calculated and displayed for each image in turn.
3. The total processing time and processing time per image are displayed.
4. An example is shown in figure 10 for 4 input images.

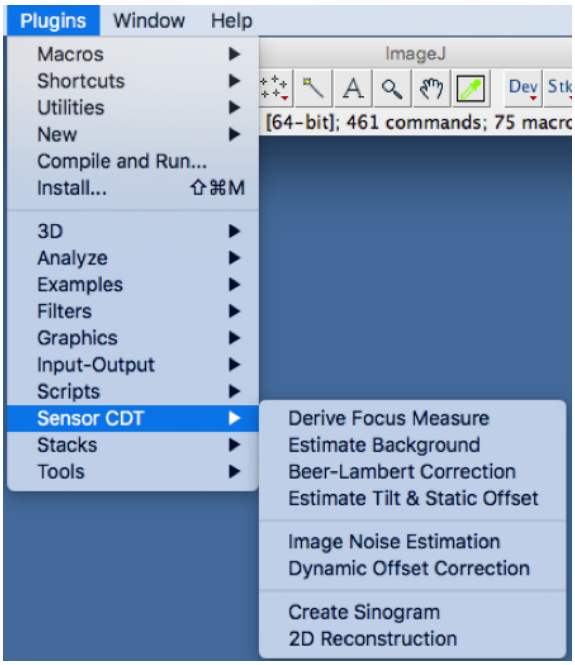

Figure 7: The OptiJ Plugin Library.

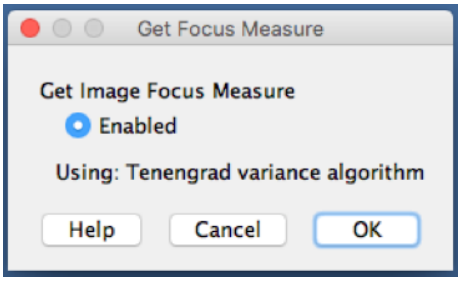

Figure 8: Derive Focus Measure Plugin GUI.

1.4 Create Sinogram Plugin

The Create Sinogram Plugin can be used as a stand-alone program for any input projections. The plugin has a simple GUI (figure 11) expects a stack of projections and will output a stack of sinograms. The user is informed of all the processes that are taking place via an ImageJ log as shown in figure 12.

1.4.1 Using the Plugin

1. Drag and drop the projection stack into ImageJ (figure 13).
2. Run the *Create Sinogram* plugin from the ImageJ *Plugins* tabs under the *OptiJ* group as shown in figure 7.
3. The plugin expects that each row in an intensity image (projection) is the intensity pertinent to a single 2D horizontal slice. If this is not the case, please rotate the input stack as required using the transform tool (under the image tab) in ImageJ.
4. Enable any additional features, these include: correction for static offset and lateral tilt (see section 4.4.5 in [1] for more details) through shifting and rotating the projections before processing is started. A *'display corrected projections'* option and a *'hold'* option allow the user to crop the projections before processing is re-started are available. The correction values can be estimated by using the *Estimate Tilt & Static Offset Plugin* in the Library.
5. Press ok to start.
6. The Plugin will output a separate stack of sinograms when completed (figure 14).

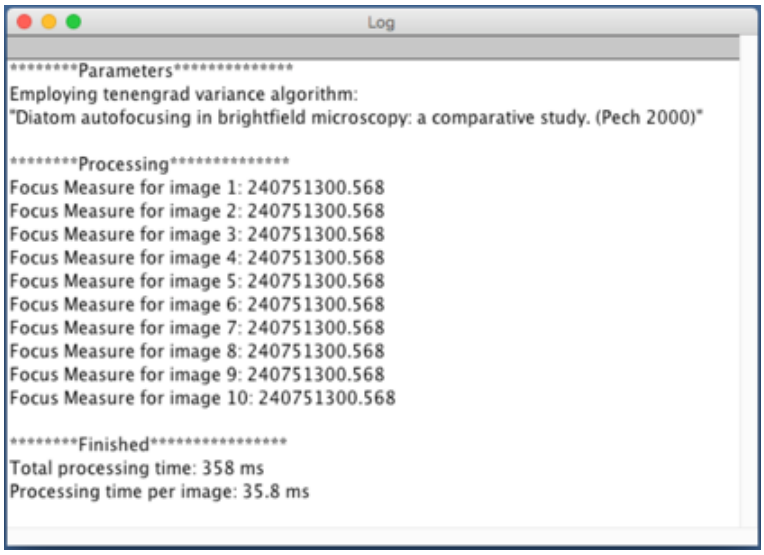

Figure 9: Derive Focus Measure Plugin logs - give a complete overview of the processing steps taking place.

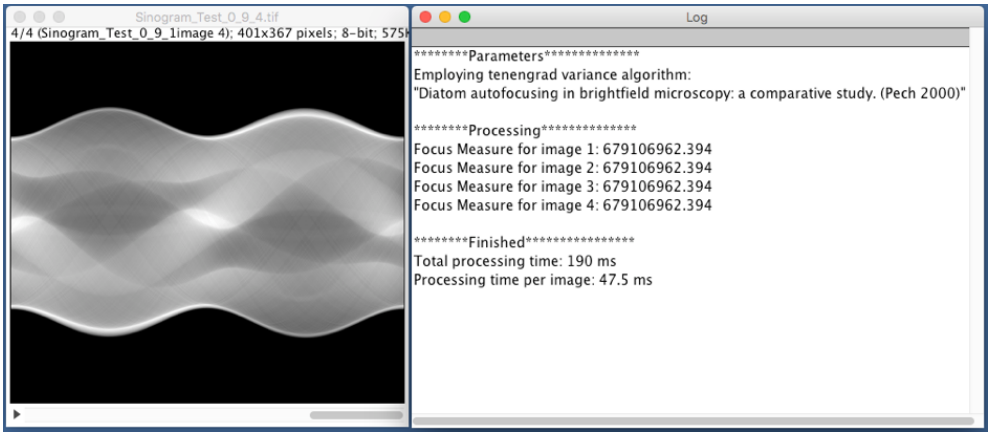

Figure 10: Using the Derive Focus Measure Plugin on a stack of 4 images. The image on the left is the input image stack, the logs on the right are the output.

1.4.2 Processing Sequence

1. User parameters are first confirmed and displayed, these include: the stack size; and any enabled additional features.
2. The processing stage then takes over:
   (a) The plugin first reads the input stack into a 3D array.
   (b) Corrections are performed if enabled and the 3D array is permuted so as to create the required sinogram stack.
   (c) The permuted array is converted to actual images.
   (d) The images are stacked creating the final sinogram stack.
   (e) Processing is ready and the sinogram stack is displayed.
3. The total processing time (from the time the plugin is executed till the stack is displayed) is logged.
4. The time taken to create the final stack is also displayed.

1.4.3 Example

Figure 14 shows a successfully executed instance of the Create Sinogram plugin with the input projection stack on the left, and the resulting sinogram stack on the right.

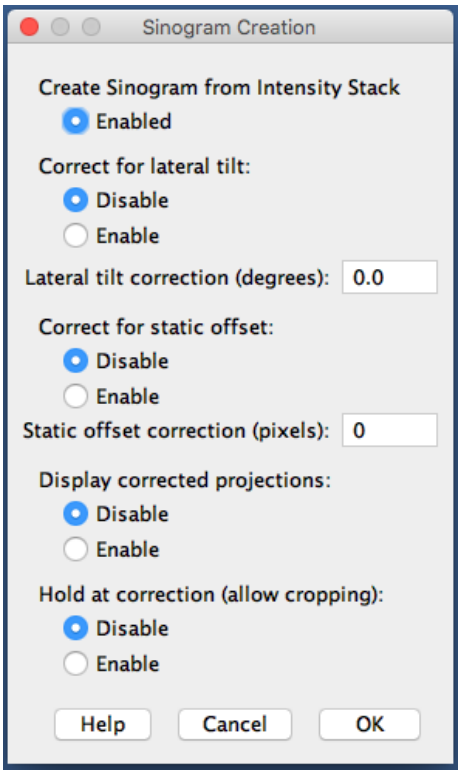

Figure 11: Create Sinogram Plugin GUI.

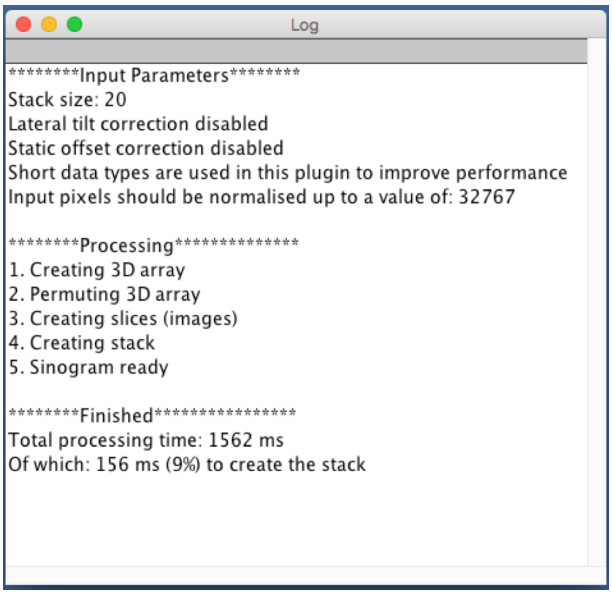

Figure 12: Create Sinogram Plugin logs - gives a complete overview of the processing steps taking place.

1.5 2D Reconstruction Plugin

2D Reconstruction is the process of creating a 2D image from a sinogram. The 2D Reconstruction Plugin can be used as a stand-alone plugin for any input sinogram/s and will output the corresponding reconstructed 2D slices using back-projection (BP) or filtered back-projection (FBP).

1.5.1 GUI and Features

The 2D Reconstruction GUI is as shown in figure 15. It has a number of inputs and features that the user can enable or disable, namely:

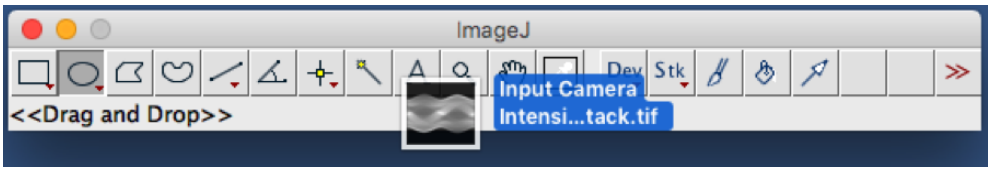

Figure 13: Using Create Sinogram Plugin - Simply drag and drop the required projection stack into ImageJ, enable the required features and press ok.

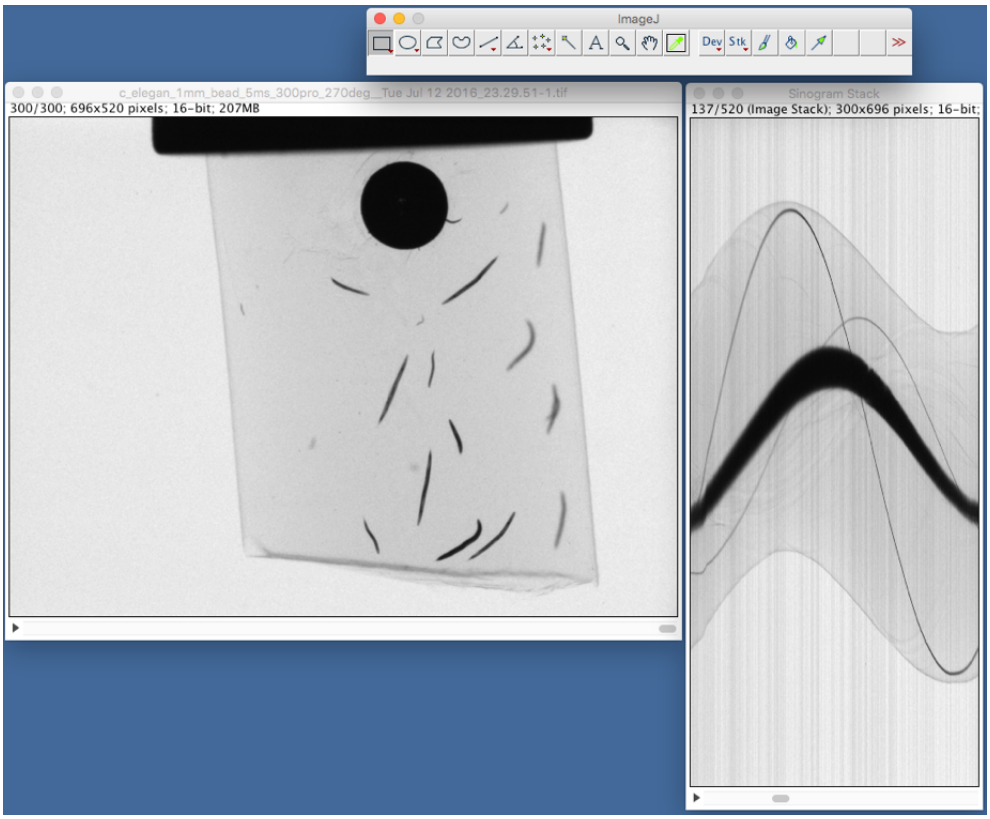

Figure 14: An example of a successfully executed instance of the Create Sinogram plugin. The input camera intensity stack is shown on the left, with the reconstructed sinogram stack on the right.

1. *Angle between projections*
   The angle between projections defines the step size that the sample has been rotated in between every projection acquired by the OPT device. This is the only user input that the plugin requires. It is important that this angle is correctly entered. The plugin will automatically calculate the number of projections taken and the angle turned from this value and the input sinogram/s. A typical value for this angle is 0.9^0 resulting in 400 projections taken in a 0-360^0 rotation.

2. *GPU/CPU acceleration*
   This is a feature which can be disabled or activated as required. If disabled, images are processed in series, if activated images are processed in parallel therefore drastically decreasing the processing time. If activated, the plugin will automatically query the host device driver and get a list of all the available GPUs (or CPUs). By default, the first available GPU (or CPU) that the driver returns is chosen. The user is informed of this via the ImageJ log. This process is fully automatic and transparent to the user. This is not platform dependent and should work with any GPU (both integrated and discrete) released from 2011 onwards. Note: As a safety feature, the plugin has a memory usage limit parameter which can be manually set. To protect integrated GPUs, popular operating systems will shut down applications which are hogging all the memory, or display blank/black results. The plugin informs the user of the memory being used.

3. *Sinogram/s angle parameter*
   By default the plugin expects the sinogram x-axis to be the angle parameter (figure 18), if this is not the chase please choose accordingly.

4. *Sinogram parameter*
   This feature will invert the sinogram intensity values if required.

5. *Filter type for FBP*
   This tab defines the filters that are available for FBP reconstruction. If FBP is enabled and no filter is chosen, this will be equivalent to BP.

6. *Reconstruction algorithm*
   There are two reconstruction algorithms available: BP, and FBP.

7. *Device Info*
   This button will output all the CPU and GPU devices that OpenCL can find on the user's platform/s.

Once the plugin is in operation, the user is kept informed of the progress taking place via an ImageJ log as can be seen in figure 16.

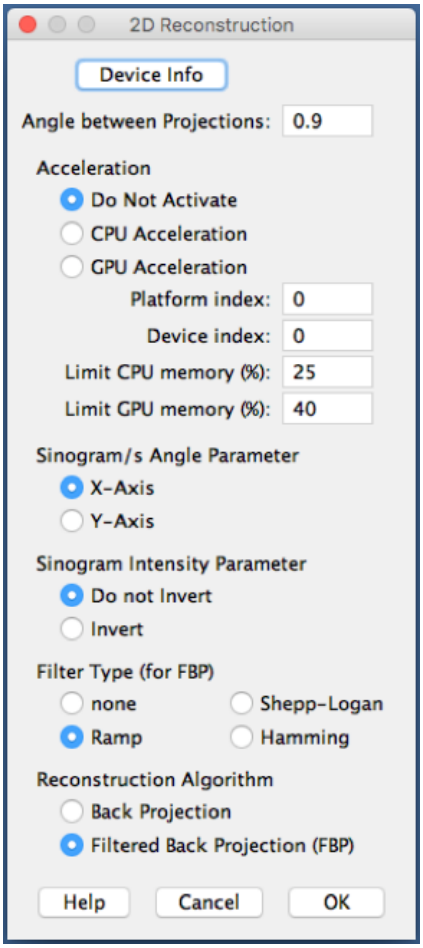

Figure 15: 2D Reconstruction Plugin GUI.

1.5.2 Using the Plugin

1. Drag and drop the input sinogram stack into ImageJ.

2. The plugin accepts both single images and stacks.

3. Run the 2D Reconstruction plugin from the ImageJ Plugins tabs under the OptiJ group as shown in figure 7.

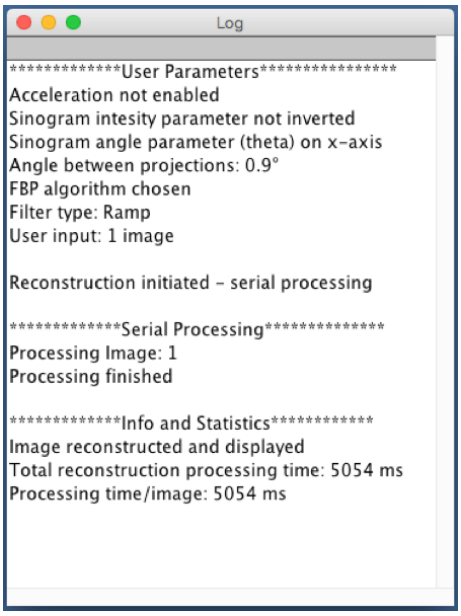

Figure 16: 2D Reconstruction Plugin logs - give a complete overview of the processing steps taking place.

4. Input the angle between projections (eg: a typical angle used in OPT experiments is 0.9^0, resulting in 400 images per 0-360^0 rotation. This is calculated automatically by the plugin and need not be specified).

5. Enable the required features and reconstruction algorithm type, a detailed explanation of all of these is given in section 1.5.1.

6. Press ok to start the process.

7. The plugin will output a separate stack of reconstructed 2D images when completed (figure 18).

1.5.3 Processing Sequence

The processing chain, as can be seen from figure 16, is displayed via an ImageJ log whilst the plugin is in operation:

1. The user input parameters and enabled features are first displayed/confirmed under the *User Parameters* tab.

2. If the *Acceleration* feature is disabled the input image/s are processed in series (figure 16).

3. If the *Acceleration* feature is enabled, the plugin automatically queries the host device driver for a list of available GPU/CPU and uses the first returned GPU/CPU to process the input image/s in parallel, if not manually set. The user is informed of the GPU/CPU and memory being used (figure 17).

4. The user is kept informed of the image currently being processed (as per the selected inputs and features, described in section 1.5.1) via the *Parallel or Series Processing* tabs in the log.

5. Once the reconstruction has been completed the user is informed and the image/s are displayed.

6. The total reconstruction processing time (time to process all images, and create the required images) is displayed, as in the reconstruction time per image.

1.5.4 Example

Figure 18 shows a successfully executed instance of the 2D Reconstruction plugin with the input sinogram stack on the left, and the resulting 2D image stack on the right.

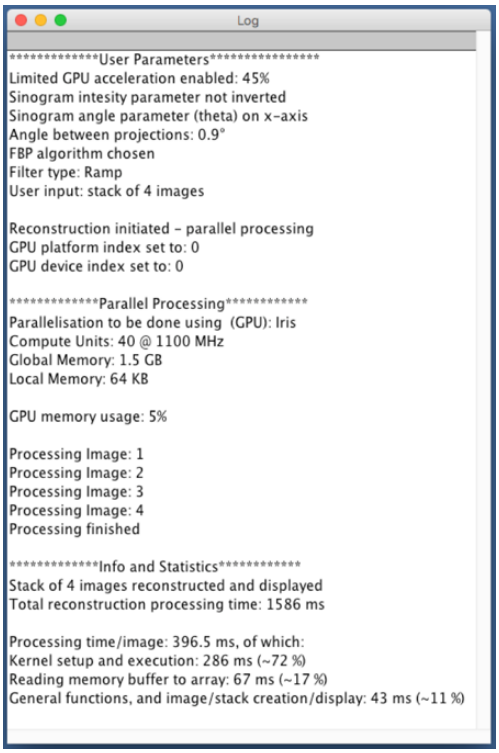

Figure 17: 2D Reconstruction Plugin log with GPU acceleration activated.

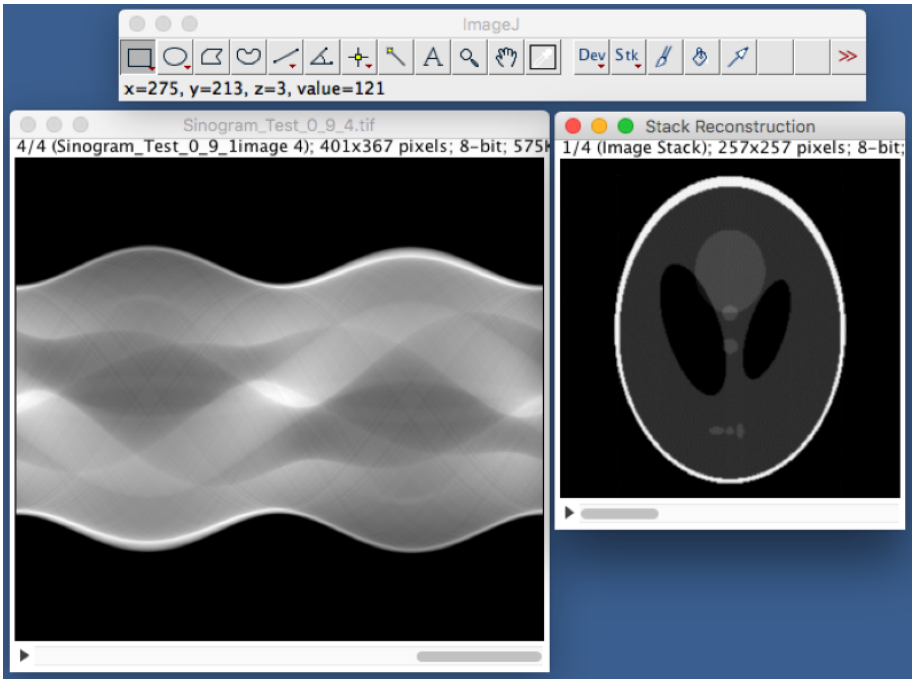

Figure 18: An example of a successfully executed instance of the 2D Reconstruction plugin. An input stack of 4 sinograms is shown on the left, with the reconstructed and displayed 2D image stack on the right.

1.6 Estimate Background Plugin

The Estimate Background Plugin takes a stack of background images as an input, averages them, and outputs a single background image. This is a very fast way to low pass filter the background. The GUI and output logs are as shown in figures 8 and 9 respectively.

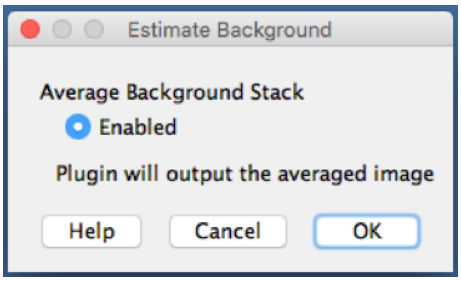

Figure 19: Estimate Background Plugin GUI.

Figure 20: Estimate Background Plugin logs - give a complete overview of the processing steps taking place.

1.6.1 Using the Plugin

1. Drag and drop the background image stack into ImageJ.
2. Press ok to start.

1.6.2 Processing Sequence and Example

1. The plugin first displays the stack size.
2. Each image is read in turn and processed.
3. When processing completes, the final average background image is displayed.
4. The total processing time is displayed.
5. An example is shown in figure 21 for 4 input images.

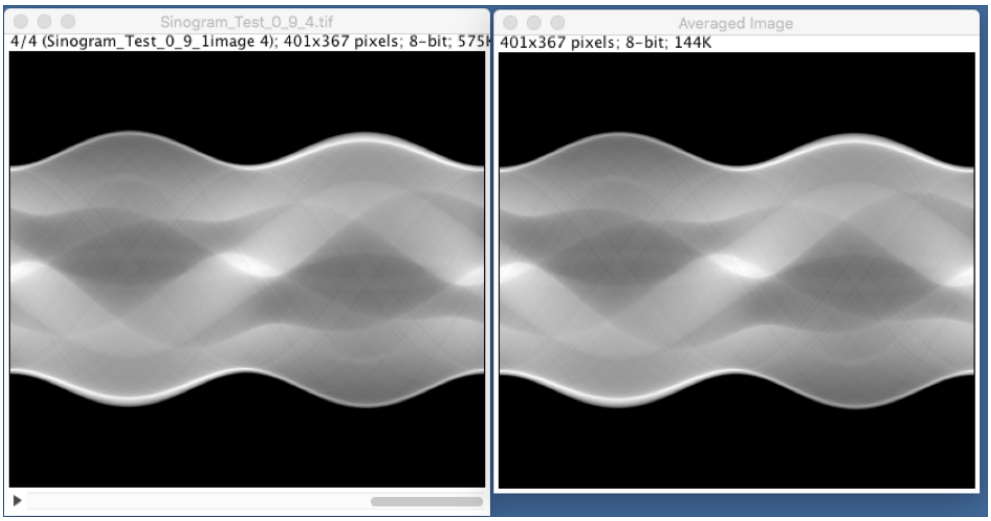

Figure 21: Using the Estimate Background Plugin on a stack of 4 images. The image on the left is the input stack, the output averaged background is on the right.

1.7 Beer-Lambert Correction Plugin

The Beer-Lambert Correction Plugin takes a projection (or stack of projections) and a background image (or stack of the same size as the projection stack), applies the Beer-Lambert law and outputs the correct projection/s. The GUI and logs are as shown in figures 22 and 23 respectively.

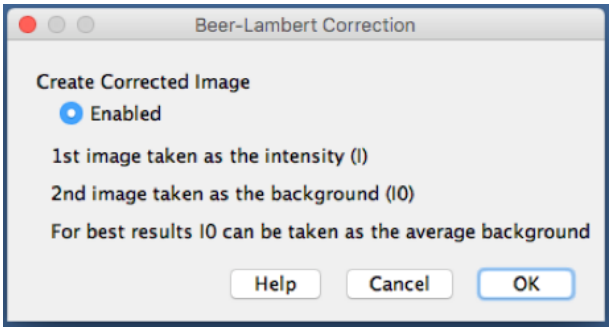

Figure 22: Beer-Lambert Correction Plugin GUI.

1.7.1 Using the Plugin

1. Drag and drop the projection/s into ImageJ.

2. Drag and drop the background image/s into ImageJ.

3. Press ok to start.

1.7.2 Processing Sequence

1. The plugin first displays which image/s are being used as the Intensity Projections, and which are being used as the Background.

2. Images are processed in turn.

3. When processing finishes, the corrected image/s are displayed.

4. The total processing time and processing time per image are displayed.

Figure 23: Beer-Lambert Correction Plugin logs - give a complete overview of the processing steps taking place.

1.7.3 Example

Figure 24 below shows an example of a single projection image before correction, and the resulting (Beer-Lambert) corrected image. This has the effect of taking into account the non uniform camera pixel intensities and variations in the illumination (see section 4.4.6 in [1]). Note that the input background image/s are not being shown.

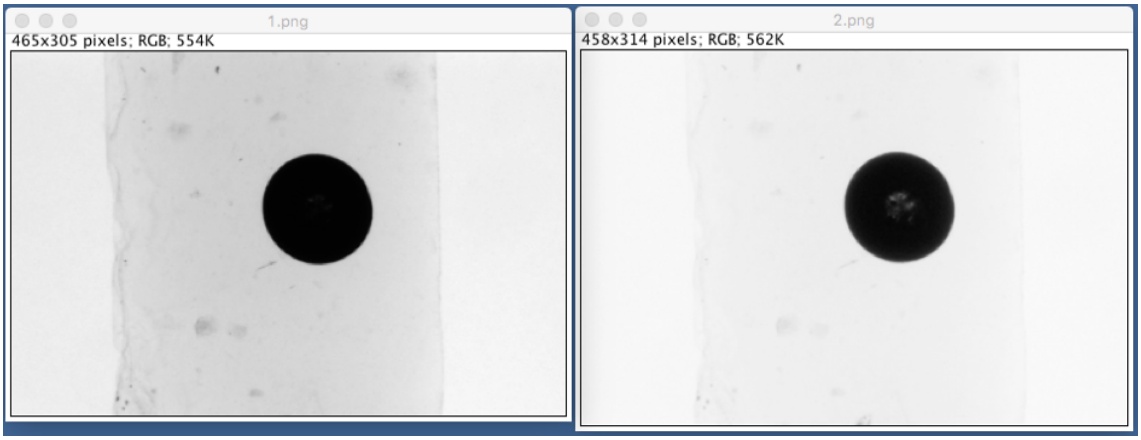

Figure 24: Comparing a projection before and after using the Beer-Lambert Correction Plugin. The image on the left is the raw projection, the image on the right is the corrected image. The average background image is also used as an input (not shown here) together with the raw projection/s.

1.8 Estimate Tilt & Static Offset Plugin

The Estimate Tilt & Static Offset Plugin is used to track the centre of mass of a marker bead embedded in the sample. It then tries to fit an ellipse to these points before attempting to estimate the following hardware errors: Static Offset; Lateral Tilt; Axial Tilt (see section 4.4.1 in [1] for a full mathematical explanation). The plugin therefore takes an image stack as an input (including the marker bead), and will output the 3 estimated hardware errors. Further features, resulting in further outputs, can be enabled and are explained below. The GUI and logs are as shown in figures 25 and 26 respectively.

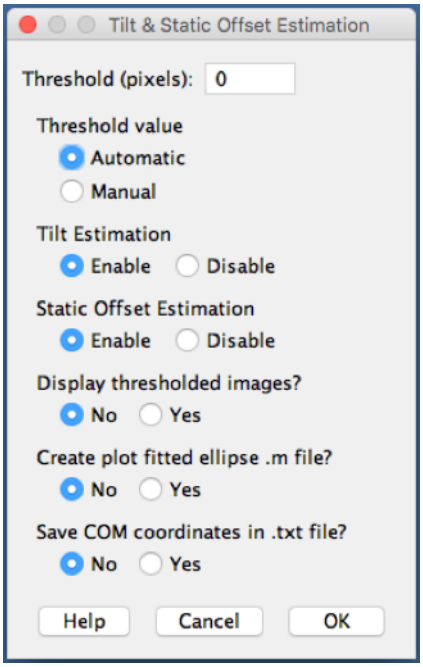

Figure 25: Estimate Tilt & Static Offset Plugin GUI.

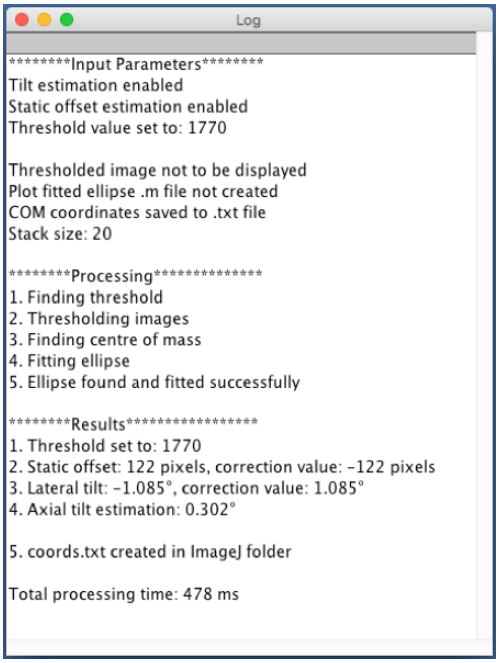

Figure 26: Estimate Tilt & Static Offset Plugin logs - give a complete overview of the processing steps taking place.

1.8.1 Using the Plugin

1. Input the projection stack into ImageJ.
2. Set the Threshold value. This is used to threshold the image such that only the marker bead remains. This value should be slightly greater than the pixel values of the marker bead, it is assumed that no such other features lie within the same pixel range. If so, please crop the image accordingly. If left as automatic, the plugin will search for the lowest pixel value in each image, average it and multiple the value by 1.5.
3. Enable or disable tilt estimation.
4. Enable or disable static offset estimation.
5. Enable or disable display threshold images. This will display the threshold image, and is a good check to see if the selected threshold value is correct.
6. Enable or disable automatic m file generation. This is saved in the ImageJ folder, and will display the tracked centre of mass coordinates and the fitted ellipse when run.
7. Enable or disable save coordinates to txt file. This will save the tracked coordinates to a text file in the ImageJ folder. This must be enabled for Dynamic Offset Plugin to operate correctly.

1.8.2 Processing Sequence

1. User input parameters and enabled features are first displayed.
2. The projection/s are first thresholded using either an automatic threshold or the manually inputted one.
3. The centre of mass of the marker bead is found (if available).
4. If enabled, the threshold image/s is displayed.
5. An ellipse is fitted to the centre of mass coordinates (if available).
6. The errors are calculated.
7. Processing finishes.
8. The errors are displayed.
9. If enabled, the centre of mass coordinates are saved to a .txt file in the users ImageJ folder.
10. If enabled, an automatic m file is generated and saved to the users ImageJ folder.
11. If Centre of mass coordinates cannot be found, or ellipse cannot be fitted (or is better approximated by a hyperbola or a parabola) the user is informed.

1.8.3 Example

Figure 27 shows and example of the Plugin tracking the centre of mass (COM) of a marker bead placed in the sample (left). Shown next to it (right) is a figure showing the tracked COM coordinates and the fitted ellipse. This is created by the m file automatically generated by the Plugin (feature needs to be enabled). The actual outputted errors are as shown in figure 26.

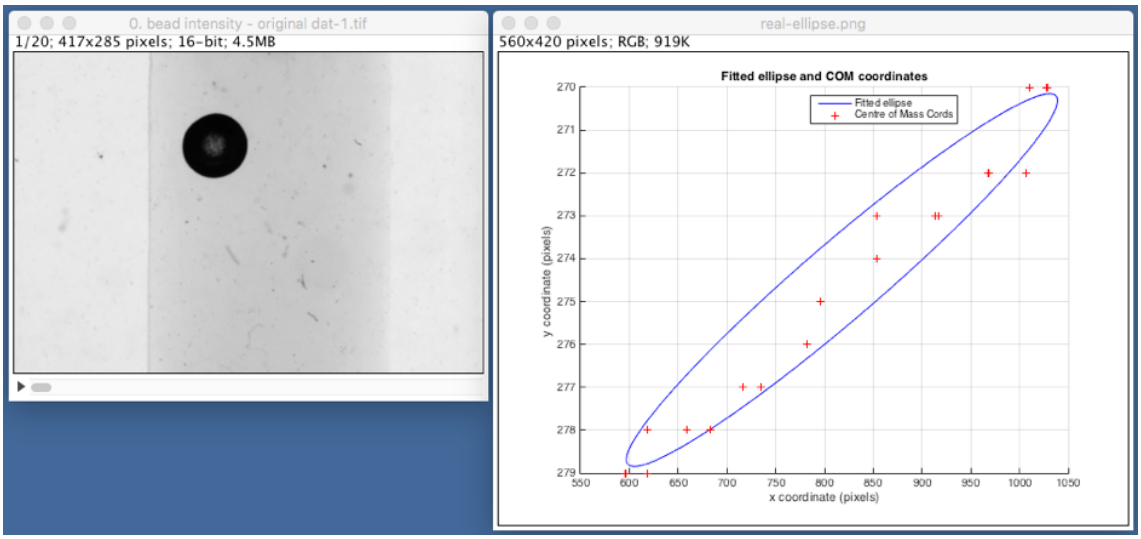

Figure 27: Using the Estimate Tilt & Static Offset Plugin. The image on the left is the input projection stack with a marker bead. The image on the right is the figure showing the tracker marker beads COM and fitted ellipse, created by the automatic m file generated by the Plugin.

1.9 Image Noise Estimation Plugin

The Image Noise Estimation Plugin estimates the noise variance in an image (or images) by sampling 100 random pixel blocks from the image and averaging. The final averaged noise variance is displayed. The GUI and logs are as shown in figures 28 and 29 respectively.

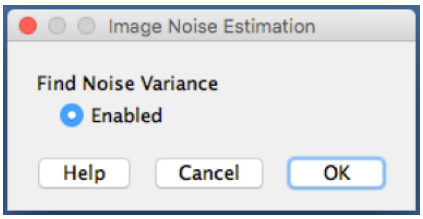

Figure 28: Image Noise Estimation Plugin GUI.

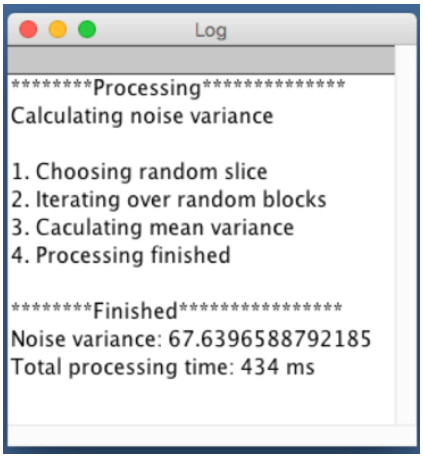

Figure 29: Image Noise Estimation Plugin logs - give a complete overview of the processing steps taking place.

1.9.1 Using the Plugin

1. Drag and drop the image/s into ImageJ.
2. Press ok to start.

1.9.2 Processing Sequence and Example

1. A random slice is chosen from the input images.
2. 100 random pixel blocks are chosen throughout the slice.
3. The variance is calculated at each pixel block.
4. The mean variance is calculated.
5. Processing is finished.
6. The mean variance is displayed.
7. The total processing time is displayed.
8. An example is as shown in figure 29.

1.10 Dynamic Offset Correction Plugin

The Dynamic Offset Correction Plugin takes a sinogram stack as an input. It also requires the coordinates of the tracked centre of mass of the marker bead from the Create Sinogram Plugin. The *save coordinates to .txt file* feature should therefore be enabled in the Create Sinogram Plugin.

The Dynamic Offset Correction Plugin fits an ideal sinusoid to these coordinates and calculates correction values as the difference between the original coordinates and this ideal fitted sinusoid. Using these correction values, the rest of the sinogram/s are corrected for dynamic offset, which is caused by wobble in the hardware. The GUI and logs are as shown in figures 30 and 31 respectively.

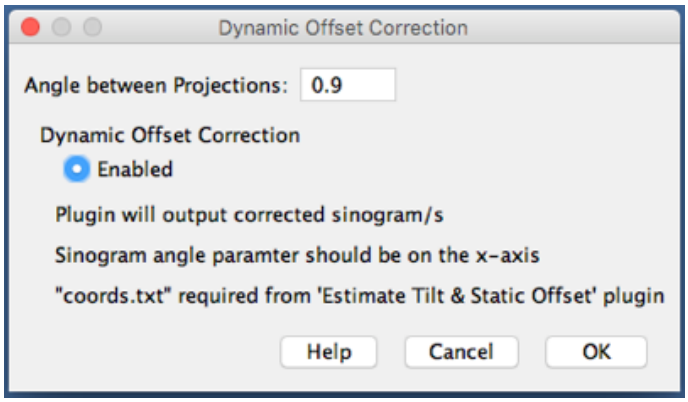

Figure 30: Dynamic Offset Correction Plugin GUI.

1.10.1 Using the Plugin

1. Drag and drop the image/s into ImageJ.
2. Input the angle between projections used during image acquisition with the OPT system.
3. Press ok to start.

1.10.2 Processing Sequence

1. The plugin checks that *coords.txt*, containing the required marker beads centre of mass coordinates, is available.
2. A sinusoid is fitted to these coordinates, and the difference between this ideal fitted sinusoid and the actual tracked coordinates is calculated.
3. These differences are used to correct the rest of the sinogram/s, i.e. they constitute the *correction values*.

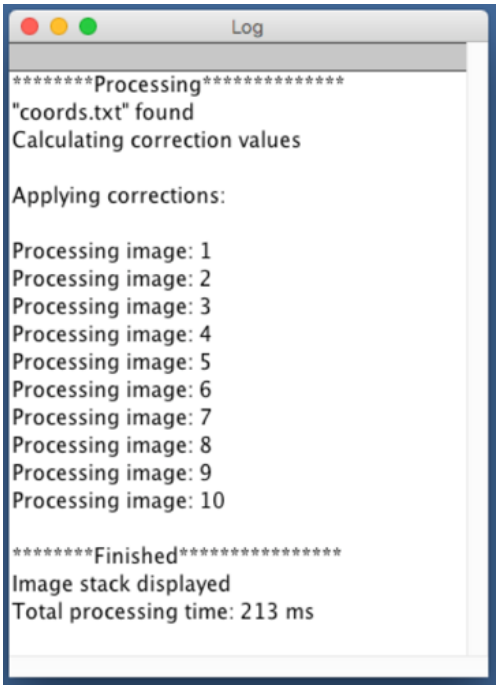

Figure 31: Dynamic Offset Correction Plugin logs - gives a complete overview of the processing steps taking place.

4. The image/s are processed in turn and the user notified through the logs.
5. When processing is finished the corrected sinograms are displayed and the total processing time displayed.

1.10.3 Example

Figure 32 shows the result of using the Dynamic Offset Correction Plugin as opposed to just correcting for static offset. The image on the left is a sinogram created with the Create Sinogram Plugin correcting only for static offset. This was then passed through this Plugin which corrects for dynamic offset error (right image).

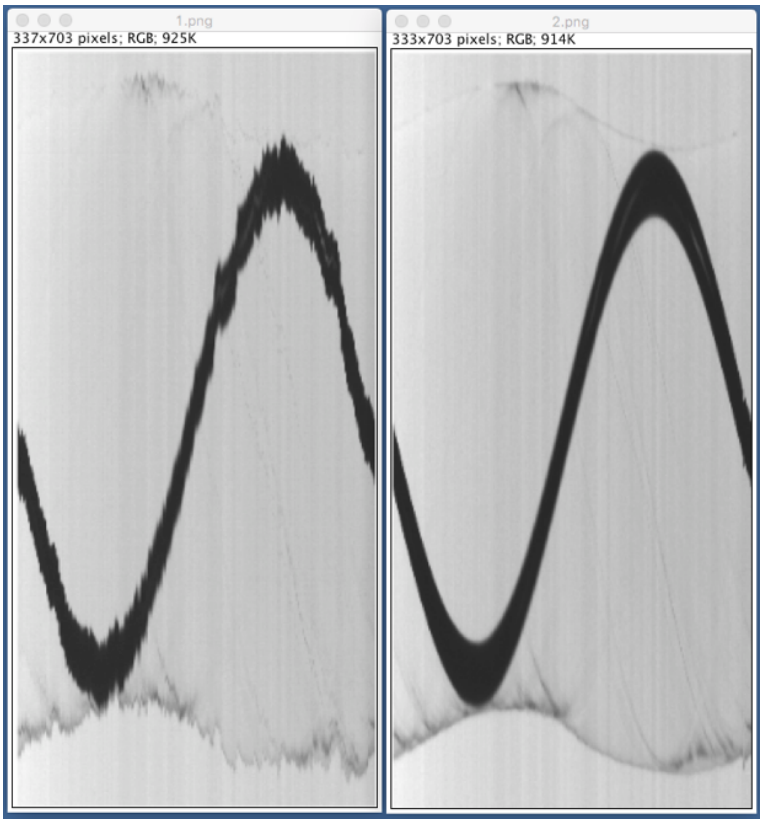

Figure 32: Using the Dynamic Offset Correction Plugin. The image on the left shows a sinogram created with the Create Sinogram Plugin which is corrected for static offset. The image on the right shows the result after adding dynamic offset correction. This requires the coordinates of the centre of mass of the marker bead to be saved in *coords.txt* in the ImageJ folder.

References

[1] J. Zammit, "Optical projection tomography: An open source imagej plugin library for image reconstruction: Sensor team challenge report," *CDT, MRES, Christ's College, Cambridge*, 24th August 2016.