User manual

by Hanna Manko

December 5, 2025

Contents

1 General description

This is a user manual for the Python-based software developed during the study **'Back-illumination Phase Imaging Enables Nanoscale Drift Stabilization in Non-ttransparent Biological Tissues'** *by Hanna Manko, Marc Tondusson, Adeline Buyreau, Morgane Meras, Stéphane Bancelin, Laurent Groc and Laurent Cognet.*
This software was developed to manage all aspects of image acquisition, analysis, and microscope control necessary for our cross-correlation (CC)-based autofocusing system. This includes the acquisition of reference images, a reference z-stack for CC, and control of acquisition with autofocusing. (Note: This is not the final version, and it will undergo some modifications with further use of the software.)

This is my first software development project. Although it has been tested on several computers in our lab, you may encounter specific problems when trying to run it on your computers and setups. Please do not hesitate to contact me if needed—I will be happy to apply any necessary modifications or improvements to make it clearer and fully reproducible. Please do not hesitate to contact me if you have any suggestions, requirements, or questions !

The graphical user interface (GUI) was created using Qt Designer for Python. The hardware was initially connected via Micro-Manager (MM) software, and MM Core was then controlled using Python libraries pymmcore and pymmcore-plus. In this manual, I will explain how to install all the required packages and use the provided user interface to perform acquisitions for those familiar with Python and MM.

1.1 GitHub repository

Together with this manual in the GitHub repository you will find:

1. the 'requirements.txt' file (the list of packages to be installed by pip)
2. the 'functions.py' file containing functions for extracting intensity, gradient, and phase images from the interferograms acquired by the QLSI module.
3. the folder "LEDcontrol" with GUI file 'LEDdisplay2.3.ui' and the python script 'LEDdisplay2.3.py' for oblique illumination setup
4. an additional folder - "Transmission_microscopeControl", containing an old software used to control commercial trans-illumination setup (NikonTi) equipped with Phasics QPI camera.
   ! Note: the code can be laggy and difficult to work with, as it has not yet been updated to the object-oriented, thread-based version. It was kept here primarily as an example of how this type of system control can be implemented at a lower level.

1.2 Setting up Micro-Manager

First, the Micro-Manager software needs to be installed. We used the Nightly Builds version of Micro-Manager 2.0.3 (20240306) https://download.micro-manager.org/nightly/2.0/Windows/. You'll need to create an MM configuration that includes all the required devices, such as the camera used for refocusing and the microscope itself, including the z-drive and Halogen lamp (**??**).
Micro-Manager device adapters have been created for most commonly used devices (compatibility can be checked on the https://micro-manager.org/Device_Support). Typically, installing drivers is required to use a new device on your computer. Once the device is connected and recognized by the computer, it should be possible to connect it using Micro-Manager.
In the transmission setup, we used the Nikon Ti lamp for illumination, which is typically listed as 'TIDiaLamp' in MM. This device has the following properties: 'ComputerControl,' 'Intensity,' and 'State,' which are referenced in the Python code. If you're using a different illumination source, you'll need to check and potentially update how these properties are defined (lines 120–124).
For example, the KURO camera was used for acquisitions in one part of this study. To connect this camera to Micro-Manager, the corresponding driver package provided by Princeton Instruments (PICAM) need to be

installed, which can be requested on their website. After installation, the camera can be connected to Micro-Manager using the PICAM device adapter in the Hardware Configuration Wizard (Arduino and Micro-Manager) (figure 2).

1.3 Arduino and Micro-Manager

The oblique back illumination setup utilized hardware triggering, which was essential for synchronizing the camera's exposure with illumination of the sample by four independent LEDs.
The LEDs were controlled via an Arduino Uno board, which was connected and managed through MicroManager. The MM was configured using several freely available sources: https://micro-manager.org/Arduino and https://github.com/nimwegenLab/MiM_NikonTi/blob/master/Docs/NikonTi_hardware_triggering.md. Generally, you should be able to configure everything based on the information provided on these two pages, although there may be some differences specific to our setup.

The Arduino Uno board was connected as shown on the figure 1. For the KURO camera the trigger cable 'Exposure Out 1' was used. The output of the cable was connected to pin 2 of Arduino board using Logic Level Converter.

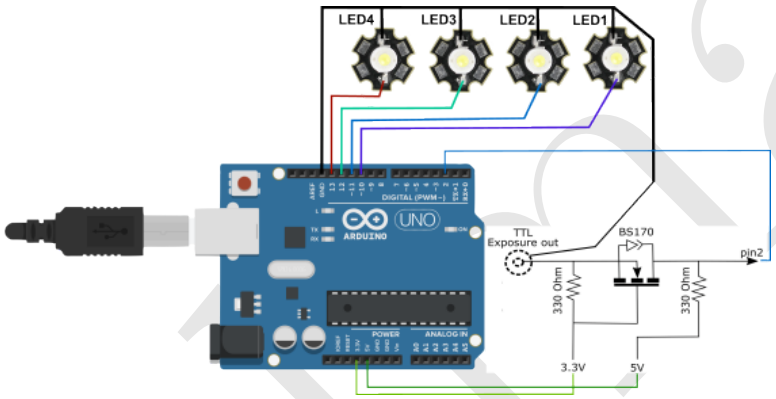

Figure 1: Arduino connection schematic.

1. Install Arduino software (https://www.arduino.cc/en/software) and connect your board to computer.
2. Ensure that your board is correctly connected (the port and the board need to be selected as shown at https://support.arduino.cc/hc/en-us/articles/4406856349970-Select-board-and-port-in-Arduino-IDE). Upload the code available at https://github.com/micro-manager/mmCoreAndDevices/blob/main/DeviceAdapters/Arduino/AOTFcontroller/AOTFcontroller.ino using Arduino software (just copy paste it into Arduino soft window and press upload) to enable the board control by Micro-Manager.
3. Go to "Hardware Configuration Wizard" (figure 2, **1.**), start creating new configuration, and connect Arduino board using "Arduino/Arduino-Hub" device adapter in MM (figure 2, **2.**).
4. Create channel group (Click on Group: "+"in window **1**, here the group was named "LEDs") and include the properties showed on window **6** figure 2 with 4 presets one for each led (figure 2, **7.**). In specific preset for each of the leds you need to choose the corresponding "Arduino-Switch-State" (In out case the states were: 4, 8, 16, 32) Then, if you want to perform acquisition with triggering in MM go to "Multi-Dimensional Acquisition" (figure 2, **8.**), enable the use of "Channels", use channel group with the leds you've created and add leds one by one using button "New".
   Now when you start the acquisition MM should acquire 4 channel images. Chose number of images by checking the "Time Points" box and entering number into "Count" box.

   ! On figure2 step 8 the exposure times for each led is set to 10 ms, you SHOULD NOT use such low exposures with Arduino triggering because it results in inability of MM to properly activate the leds. Typically exposure time of 50 ms was used on our setup which was not causing any problems (40 ms works as well, smaller values were not properly tested).

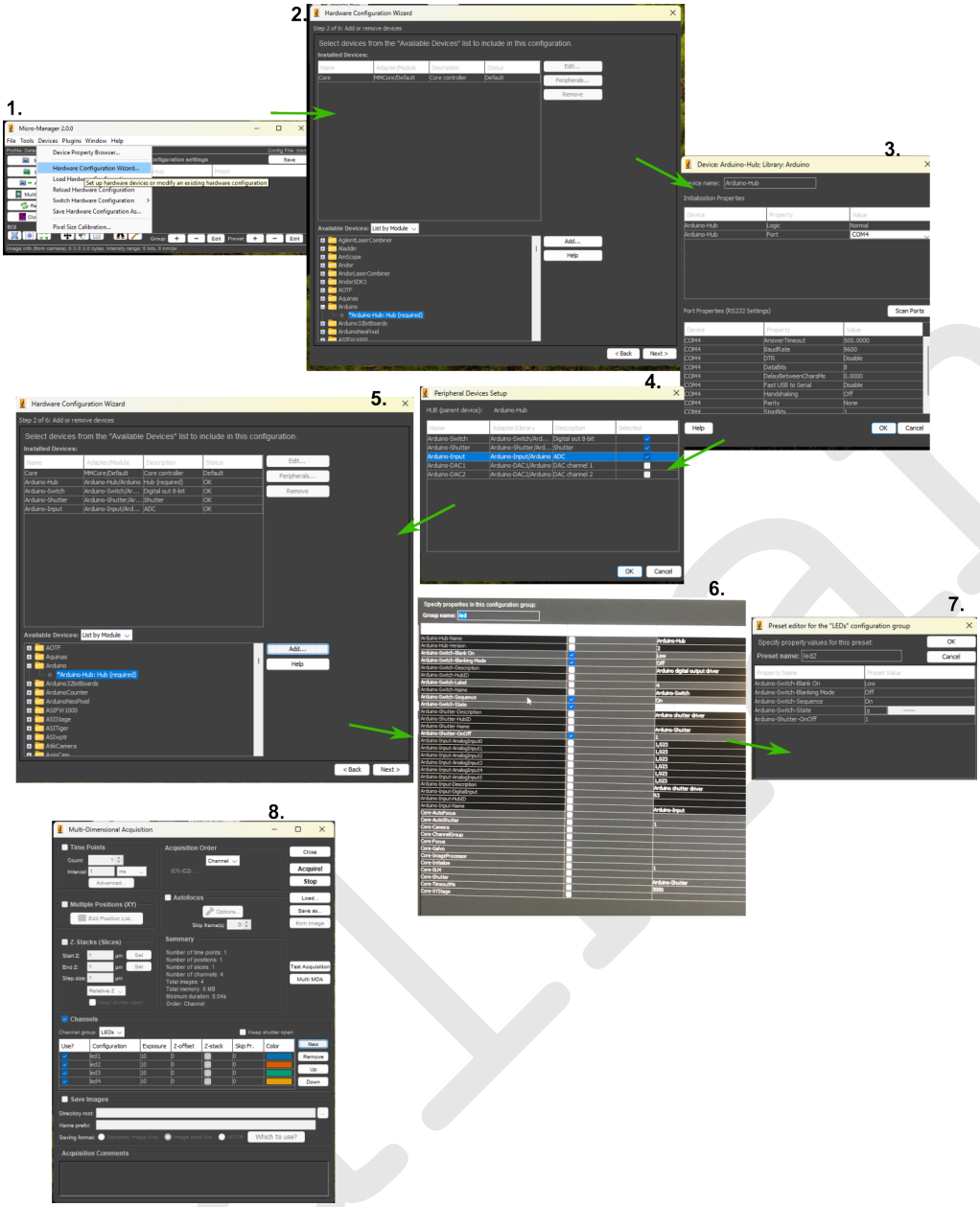

Figure 2: The connection of Arduino board to MM and setting up the control of illumination. Some intermediate steps are not shown, but usually it simply involves clicking "Next" or "OK."

1.4 Setting up Python

You can start by installing miniconda https://docs.anaconda.com/miniconda/. Open Anaconda/Command prompt on your computer (you can do this by typing its name in search box in Windows). It is recommended to use a separate Python environment to avoid package conflicts. To create the environment with specified version of python run the command:

```
conda create -n Environment_name python=3.9.18 # final version was not tested with other
                                                 versions of python
```

We used Python 3.9.18. Once the environment is activated, install the required packages from the 'requirements.txt' file available in the GitHub repository.

```
pip install -r /path_to/requirements.txt
```

Before running the script some modifications will be required (need to be made only once):

1. Open LEDdisplay2.3.py with your preferred code editor.
2. Insert path to Micro-Manager folder on your computer (typically line 102).
3. Specify the name of the configuration file '*.cfg' (line 103)
4. Go to the end of the script where the GUI window is defined and put the correct path to the "Autofocus.ui" file (line 920)
5. Save the modified script

All necessary functions are defined within the main code. You can follow the comments and adjust certain values if needed for your specific setup. However, it's recommended not to modify any parameters until you've confirmed that everything is functioning correctly.
Now you can run the script from your Anaconda or Command prompt:

```
python /path_to_file/LEDdisplay2.3.py
```

or by running in in your preferred Python IDE (for example Spyder, PyCharm, etc.)
Note: The code is designed to automatically load the most recently saved file with the appropriate name whenever files like reference images or reference z-stacks.

1.5 Working with GUI for oblique back illumination setup

1. Insert or browse to path to the designated save folder (figure 3, yellow box). Optionally, you can specify file name.
2. You have the option to choose whether to use all four LEDs for illumination while viewing the live feed or to select a specific LED. It is advisable to check the illumination provided by each LED and adjust the fiber positions around the objective if necessary, just prior to acquisition (figure 3, blue box).
   ! Note: sometimes Arduino board can give errors while controlling it through MM. It might be required to run ones the live feed with one of the LEDs activated before proceading to further steps.
3. Then the reference images need to be acquired. In our case (piezo nanopositioner with range of 100 μm) we use by default the z-stack in the range of -45:45 μm with step of 10 μm (figure 3, red box). These images are further averaged and saved.
4. Further the reference z-stack need to be recorded (figure 3, green box).
5. In order to start the acquisition with active autofocusing, enter the number of images and the autofocusing frequency (figure 3, purple box).

It is possible also to control how many LEDs are used for illumination, indicated as illumination pattern in GUI. With 2 LEDs illumination one can view DPC images. Normally, in this case the illumination pattern will consist of 2 opposit LEDs "1" and "2".

The hDPC (or DPC) images can be visualized in real time using "hDPC Live" button.

Additionaly, it is possible to start simple acquisition with chosen illumination pattern. In this case the stacks of images will be automatically saved into specified folder. Moreover, the stacks of images will be automatically separated into substacks of max 2Gb size (this can be modified in main code).

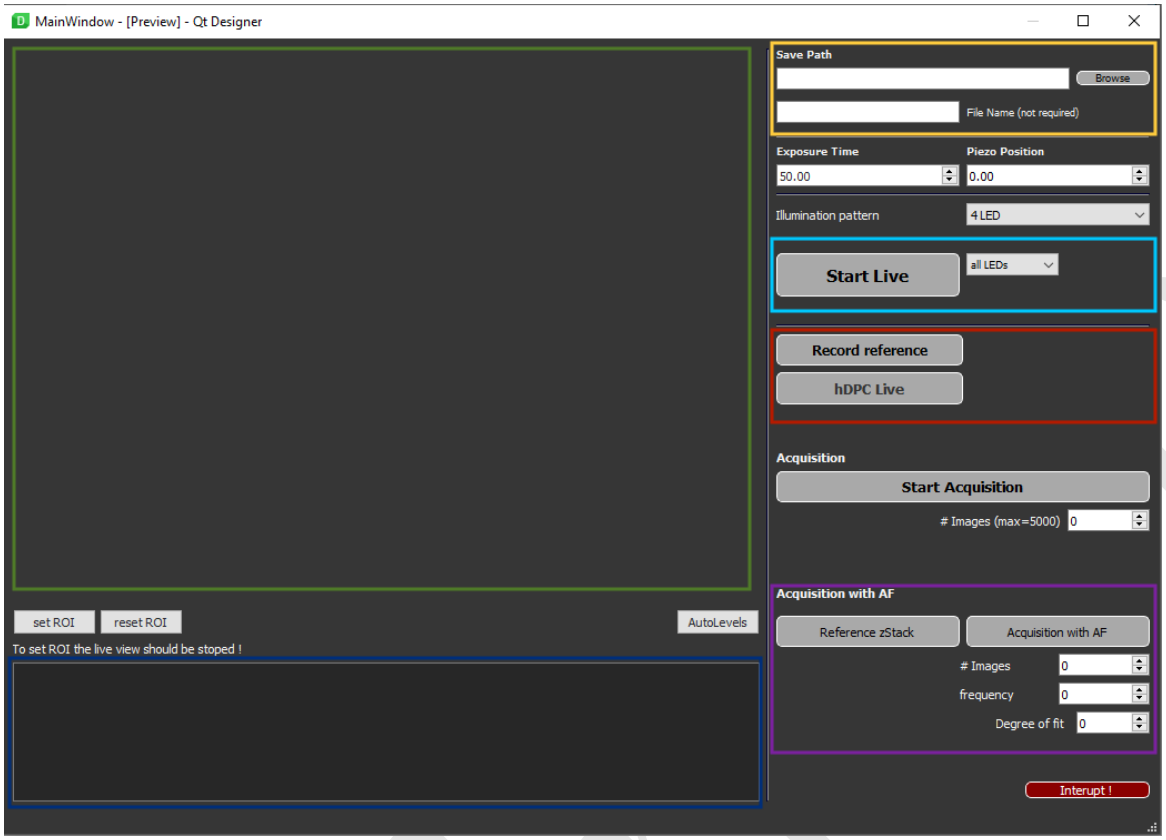

Figure 3: GUI window for oblique illumination setup

1.5.1 Handling the data

The acquired stacks of are saved as *.tif files so tifffile library can be used to upload then in python. While using led illumination the images acquired sequentially providing 4 channels. First we need to upload the reference images.

```
from tifffile import imread
import numpy as np
import matplotlib.pyplot as plt
ref = imread('path_to_file/reference.tif')
```

Further the differential contrast images are obtained as:

```
images = imread('path_to_file/Stack.tif') # uploading the stack of images
def DPC(im1, im2): # defining the function to calculate differential image
  dpc = (im1-im2)/(im1+im2)
  dpc_mean = DCP.mean()
  dpc_std = DPC.std()
  DPC = np.clip(dpc, dpc_mean-4*dpc_std), dpc_mean+4*dpc_std)
  return DPC
Idpc1 = DPC(images[:,3]/ref[3], images[:,0]/ref[0]) # the order of images might change
                                                    depending on your setup
Idpc2 = DPC(images[:,1]/ref[1], images[:,2]/ref[2])
hDPC = Idpc1-Idpc2 # the combination image
```

1.6 Additional codes that may be useful

To build the graph showing simulteneously the drift calculated with its histogram on the side and the z-drive position value (as it was shown in the main article):

```
import matplotlib.gridspec as gridspec
fig_drifts = plt.figure(figsize=(10,7))
color = 'tab:blue'
gs = gridspec.GridSpec(4,4, wspace = 0.6)
ax_main1 = plt.subplot(gs[1:4, 0:3])

ax_main1.plot((drift[:,1]),':o', markersize=5, color=color)
ax_main1.set_xlabel("Refocusing step", fontsize=18)
ax_main1.set_ylabel("Position of z drive,  $\mu$m", fontsize=18, color=color)

ax_main2 = ax_main1.twinx()
color = 'tab:red'
ax_main2.plot(drift[:,0], ':o', markersize=5,color=color)
ax_main2.set_ylim((-0.8,0.8)) # limiting the range of axis for the calculated drift
ax_main2.set_ylabel('Calculated drift, $\mu$m', rotation=270, fontsize=18, labelpad=15, color=
                                                color)
plt.grid()

ax_yDist = plt.subplot(gs[1:4, 3],sharey=ax_main2)
ax_yDist.hist(drift[:,0],bins=5,orientation='horizontal',align='mid', color='r')
ax_yDist.yaxis.set_ticks_position('right')
ax_yDist.set(xlabel='count')
ax_yDist.set_ylim((-0.8, 0.8))
plt.show()
```

The code to obtain the drift for the stack of images you will need to upload reference image, z-stack to be used for cross-correlation and the stack of images for which you want to determine the axial positions (before running full loop check for one image if everything is going well):

```
import numpy as np
from numpy import where
import silx
from silx.image import sift
import sys
sys.path.append('path_to_functions.py_file') # to enable importing functions from another file
                                               located at this path
from functions import norm_data, ncc # if required you can just copy these functions from '
                                               functions.py' file

pos = np.loadtxt('path_to_file/zStack_posRed.txt')
drift = []
poss = pos-pos.min()
x = poss-poss[int(len(poss)/2)]
xx = np.arange(x.min(), x.max(), 0.001)
sift_ocl = silx.image.sift.SiftPlan(template=zStack[int(len(zStack)/2)], device = "GPU")
keypoints = sift_ocl(zStack[int(len(zStack)/2))
mp = sift.MatchPlan()
for i in range(0, len(stack)):
  corrcoef = []
  im = stack[i]
  im_keypoints = sift_ocl(im.astype(np.float32))
  match = mp(keypoints, im_keypoints)
  sa = sift.LinearAlign(zStack[int(len(zStack/2))])
  for i in range(0, len(zStack)):
    corrcoef.append(ncc(im[50:550, 50:550], zStack[50:550, 50:550]))
  y = np.array(corrcoef)
  model2 = np.poly1d(x, y, degree)
  drift.append(xx[where(model2(xx) == model2(xx).max())])

plt.plot(x,y) # to show the last cross-correlation curve
plt.plot(x, model2(x)) # to plot the fit
plt.plot(drift, '+:') # to plot the calculated drift
```

1.7 Throubleshooting

1.7.1 Algorithm failing

One of the most common reasons of algorithm failure is problems with xy-drift correction. If lateral drift was too strong and was not corrected mechanically, the SIFT method will fail to find matching keypoints. In this case the acquisition loop will be automatically interrupted with saving the acquired data. Furthermore, in the case of significant shift in z during the time between reference z-stack acquisition and the beginning of acquisition, or if the significant morphological changes occurred in the tissue, there may be almost no similarities between images anymore so SIFT will fail.

Additionally, the cross-correlation curves are shown at every autofocusing step. If the shape becomes too distorted the drift can no longer be determined with desired precision, then is it better to interrupt the acquisition and record the reference z-stack again.

...

2 Old version of code: trans-illumination setup control

2.1 Working with GUI for trans-illumination setup

If the installation and MM initialization were successful, the GUI window should appear (figure 4). It is possible that you may encounter some issues; if standard troubleshooting methods (such as searching StackOverflow) don't resolve the problem, do not hesitate to contact me).

1. Create a folder where you will save all the acquired data, and insert the path into the line (figure 4, yellow box).

2. For reference, there are two important parameters to enter: Exposure Time and Lamp Intensity. Typically, the exposure time in our experiments was set to 80 ms, and the lamp intensity was adjusted to values between 7 and 8 (depending on the sample; if the tissue was thicker, the intensity needed to be increased). To properly adjust these values, you can use the "Check max value" button. The value will be displayed in the message box of the GUI. Ensure that there is no saturation, and it is preferable to keep the value around 40,000 to 50,000 (assuming 65,000 is the maximum) (figure 4, red box).

3. To acquire the reference, enter the number of images to average (typically set to 200) and press the "Get reference" button (figure 4, green box). During the reference acquisition, move the XY stage around the sample quickly; you may also slightly adjust the focus knob. The goal is to obtain an averaged wavefront from the specific sample. A new reference must be acquired for every new sample. The obtained reference will be displayed in the Napari Viewer. Ensure that the fringes of the interferogram are clearly distinguishable (ex: figure 4, b).

4. Once the reference has been acquired, you can adjust the Exposure Time (typically set to 10 ms) and Lamp Intensity (usually between 10 and 12, depending on the sample) to the values that will be used during the acquisitions. It is important to check the maximum pixel intensity value again.

5. Now you can move around the sample to search for the area of interest.

6. Ones you defined the field of view it shouldn't be moved and the reference z-stack can be recorded. There are two possibilities: to record full z-stack, with regular stepping (by default = 0.05 μm) or to use weirdostepping (0.05 μm around 0 position in tghe rangeof [-0.4,0.4]μm, and bigger steps (0.2μm) further from it). There is also such parameter as "N im to average" (default = 10), can be changed if required (smaller will increase the speed) (figure 4, blue box).

   The default values will be used unless otherwise specified !

7. **Acquisition with refocusing.** You will need to input the 'Number of images' you plan to acquire and the 'Refocusing frequency,' which determines how many images will be captured between each refocusing step. There are two methods available for refocusing: using gradient images (as described in the article) or using only intensity images. The intensity-based option was added to allow the software to be used even when the QLSI module is unavailable. In such cases, adequate autofocusing performance can still be achieved using standard EMCCD or sCMOS cameras with intensity images (see figure 4, purple box). A graph displaying the determined drift and the cross-correlation (CC) curve at each autofocusing step will appear, enabling you to monitor the system's performance.

   Note: It is recommended to begin by launching a small test acquisition (e.g., 20 images with a refocusing frequency of 5) to ensure that everything is functioning correctly. If errors occur, such as an unusual CC curve or erratic drift values, this could indicate significant sample drift, causing the field of view to shift from the one used to record the z-stack. In such cases, re-recording the z-stack in the current field of view is advised.

   ! The acquisition process can be stopped by pressing the 's' key on your keyboard (it may require multiple presses). In this case, the recorded data will be saved. Alternatively, pressing the 'q' key will interrupt the acquisition without saving any data.

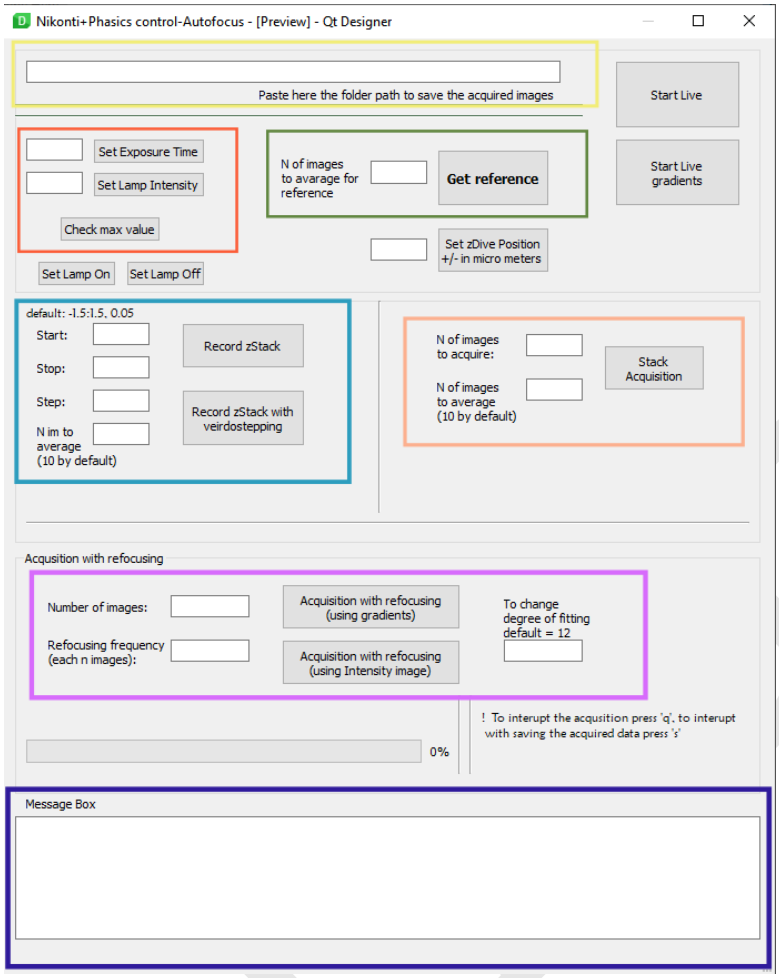

Figure 4: GUI window for transmission illumination setup

8. There is also an option to record a stack of raw images (interferograms in the case of the QLSI module), which can be analyzed later. This is typically used to assess the drift present in the sample during acquisitions without active refocusing. (figure 4, orange box). It is recommended to not record too large stacks of images and to limit the acquisitions to 300 frames (*Note : for the moment the stacks are continuously saved into single file. It will be modified in near future*)

9. Important messages and descriptions of the actions being performed will be displayed in the 'Message Box.'(figure 4, dark blue box).

2.1.1 Handling the data

The recorded stacks are saved as *.npy files. The ref.npy file contains several important arrays: the reference image itself, two masks for selecting the position of harmonics in the Fourier transform of the interferogram, and two gradient images of the reference image. To load it, you can use:

```
reference = np.load('/path_to/ref.npy')
reference = reference.reshape(5, int(reference.shape[0]/5), reference.shape[1])
ref = reference[0]  # this is the reference image that was recorded
```

For example to load the acquired raw z-stack use:

```
zStack = np.load('/path_to/zStackRaw.npy')
zStack = zStack.reshape(int(zImage.shape[0]/ref.shape[0]), ref.shape[0], ref.shape[1])
```

```
zStack.shape ## will output the shape, can be (60,1500,1500) in the case of z-stack with
                                                    regular stepping
```

Then you can easily display the images using your preferred method in Python, such as with the matplotlib library.

```
import matplotlib.pyplot as plt
plt.imshow(zStack[0]) # to display first image in the z-stack
```

You can also save the stack in *.tif format, which allows it to be opened in Fiji if needed.:

```
from tifffile import imwrite
imwrite("/path_to_save/zStack.tif", zStack.astype(np.float16)) #stack can saved as 16-bit
                                                    image to prevent files from becoming too large
```

The drift value calculated at each step of axial position determination using the cross-correlation method, along with the current position read from the microscope's z-drive are saved into a 'detectedDrift.txt' file. You can load it as a NumPy array:

```
drift = np.loadtxt('path_to_file/detectedDrift.txt')
# drift[:,0] corresponds to calculated drift values
# drift[:,1] to position of the z-drive
```

An additional file, 'zStack_posRed.txt', is saved during acquisitions. This file contains the position of the microscope's z-drive during z-stack recording, with the position being read after each step. This is particularly useful for traditional microscope z-drives, like the one on the NikonTi microscope, due to potential stability issues with these drives. In some cases, the drive may fail to move to the desired position or may perform steps that are smaller or larger than intended.

Acknowledgements

Thanks to the guy with too long family name for being the first tester.