Appendix C

Interferogram and Profile analysis

C.1 Python script: Profile calculation for CSI

The extraction of a surface profile is done following the method of CSI. The whole process is discussed in Sec. 3.2. The **basic operations** are the following:

```
"""
Load interferogram files and create a stack
"""
"""
...
"""
"""
Load calibration file
"""
"""
...
"""

def gaus(x,a,x0,sigma,c):
    return a*np.exp(-(x-x0)**2/(2*sigma**2))+c

"""
PER PIXEL ANALYSIS
"""
for i in range(rows):
        for j in range(cols):
        # extract interferogram in vertical axis (normal to the surface)
            z = image_stack[:,i,j]
            zm = z - np.median(z)
            za = np.abs(zm)
        # take the "envelope"; sigma is selected as a good trade-off with visual criteria.
            zg = gaussian_filter1d(za, sigma=30)

        # make a first guess for the gaussian fitting of the "envelope"
            nmax = np.argmax(zg)
            amp = zg[nmax]

            try:
            # statistically 60 pixels is a good guess for the specific white light source used
                sigma = 60
                c = np.median(zg)
                guess = [amp,nmax,sigma,c]
                popt, _ = curve_fit(gaus,n,zg,p0=guess)
                peak = popt[1]
```

```
            # the surface elevation is the interpolated value of the
    calibration array with input
            # the calculated peak of the best-fit.
                profile[i,j] = curve(peak)
            except:
                profile[i,j] = None

"""
Save profile
"""
"""
...
"""
```

C.2 Python script: Profile calculation for SRWLI

This process is presented in Sec. E. The **basic operations** are the following:

```

"""
Load images of interference, reference, sample and background.
Load array containing the calibrated values for the decomposed
    wavelengths.
"""

# compute wavenumber array
k = 2*np.pi/lamda

# smooth the following signals before using them to isolate the
    interference cosine term
Is_g = gaussian_filter1d(Is, sigma=3, axis=1)
Ir_g = gaussian_filter1d(Ir, sigma=3, axis=1)
Ib_g = gaussian_filter1d(Ib, sigma=3, axis=1)

# calculate interference cosine term --> 'Ipr'
eps = np.finfo(np.float64).eps
nom = 2.0*np.sqrt(Is_g)*np.sqrt(Ir_g)
nom[np.where(nom==0)] = eps
Ipr = (Itot-Is_g-Ir_g-Ib_g)/nom

# EXTRA PRE-PROCESSING
# extract waviness --> 'Iprmodg2'
Ipr_g = gaussian_filter1d(Ipr, sigma=11, axis=1)
Ipr_mod = Ipr - Ipr_g
Ipr_modg = gaussian_filter1d(Ipr_mod, sigma=3, axis=1)
Iprmodg2 = Ipr_modg - np.median(Ipr_modg)

# Linear resampling (factor=1) of the wavenumber samples
num_resample = cols
if num_resample % 2 == 0:
    num_resample -= 1
k_new = np.linspace(k[0], k[-1], num_resample)

# calculate frequency range and samples
dk = abs(k_new[0] - k_new[1])
Fn = 1/(2*dk)
Nfreqs_f = num_resample
f_axis = np.linspace(0, Fn, Nfreqs_f//2+1)

# initialize profile array
z = np.zeros(rows)
```

```

"""
Iterate procedure for each row of the image
"""
for i in range(rows):
  # resampled waviness signal
    f_rsmpl = interp1d(k, Iprmodg2[i], kind='cubic')
    Ipr_new = f_rsmpl(k_new)

    # FT calculation
    fourier = np.fft.fft(Ipr_new, n=Nfreqs_f)
    fourier = np.abs(fourier)
    fourier = np.fft.fftshift(fourier)
    fourier = fourier[Nfreqs_f//2:]

    # Upsampling of fourier signal to "increase" resolution
    Nf_times = 20
    f_axisnew = np.linspace(f_axis[0], f_axis[-1], Nfreqs_f*Nf_times)
    f_rsmpl2 = interp1d(f_axis, fourier, kind='cubic')
    fourier_new = f_rsmpl2(f_axisnew)

    # smooth fourier signal to eradicate noise for peak detection
    sigma_f = Nf_times*1        #Nf_times/4
    fourier_g = gaussian_filter(fourier_new, sigma=sigma_f)
    argmax_idx = np.argmax(fourier_g)
    # calculate surface elevation
    z[i] = np.pi*f_axisnew[argmax_idx]/um
```

C.3 Python script: Piezo actuator calibration

This process is explained thoroughly in Sec. 3.6. The **basic operations** are the following:

```
"""
Load interferogram files and create a stack
"""
"""
...
"""

"""
PHASE CALCULATION FROM EACH INTERFEROGRAM
"""
phases[0] = calc_phase(image)

for j in range(1,listlen):
    image = ((io.imread(file_list[j])).astype(float))
    # odd dimensions of image --> Frequency 0 of Fourier axes is exactly
     on the center of the image
    image = im2odd_dim(image)
    phases[j] = calc_phase(image)

"""
CALCULATE DISPLACEMENT FROM PHASE SHIFTS
"""
uwphases = np.unwrap(phases)
dp = np.diff(uwphases)

# Phase difference converted to displacement of fringes.
# In michelson interferometers, displacement of the sample is equal to
# half the optical path difference
```

```
lamda = float(input('Enter the peak wavelength of the source (nm)...\n')) # in nm
displacements = dp/(2*np.pi)*lamda/2 # in nm
displacements = np.concatenate([[0],displacements]) # zero as the first position

incremental = np.cumsum(displacements)
grad = np.gradient(incremental)
grad = reject_outliers(grad, m = 2.)
dp_mean = np.mean(grad)

incremental = - incremental if dp_mean < 0 else incremental

"""
Save Calibration file
"""
...
"""
```

C.4 Python script: Profile post-processing

This process is explained in Sec. 3.3. It requires user visual feedback and for this reason a small GUI has been built. User events signal callback functions that do the basic operations of the GUI. The **callback functions** are the following:

```
"""
Create Matplotlib window with widgets for buttons, figures, axes, span selectors and ROI selectors.
"""
"""
...
"""

def line_select_callback(eclick, erelease):
    global ROI

    if toggle_selector_RS.active:
        x1, y1 = int(round(eclick.xdata)), int(round(eclick.ydata))
        x2, y2 = int(round(erelease.xdata)), int(round(erelease.ydata))
        ROI = [(x1,y1), (x2,y2)]

def enable_ROIsel(event):
    global ts_id

    del ROI[:]

    toggle_selector_RS.set_active(True)
    toggle_selector_RS.set_visible(True)
    ts_id = fig.canvas.mpl_connect('key_press_event', toggle_selector)

def toggle_selector(event):
    if event.key=='enter' and toggle_selector_RS.active:
#           print(' RectangleSelector deactivated.')
        textbox.set_val('ROI selected')
        toggle_selector_RS.set_visible(False)
        toggle_selector_RS.update()
        toggle_selector_RS.set_active(False)
        fig.canvas.mpl_disconnect(ts_id)

def onselect(xmin, xmax):
    global flag_save, thres, unrotated
```

```python

    thres[thres<xmin] = xmin
    thres[thres>xmax] = xmax
    ax1.cla()
    ax1.hist(thres.ravel(), bins=255)
    ax1.set_title('Press left mouse button and drag to select the wanted histogram range')
    ax2.imshow(thres, cmap=plt.cm.viridis)
    ax2.set_title('Thresholded profile')
    unrotated = 'thres'
    textbox.set_val('Low-high thresholds set as [%d,%d]'%(xmin,xmax))
    cbar2.set_clim(vmin=np.amin(thres),vmax=np.amax(thres))
    cbar2.draw_all()
    flag_save = 1

def crop(event):
    global thres, flag_save

    (x1,y1), (x2,y2) = ROI
    thres = thres[y1:y2,x1:x2]

    ax1.cla()
    ax1.hist(thres.ravel(), bins=255)
    ax1.set_title('Press left mouse button and drag to select the wanted histogram range')
    ax2.imshow(thres)
    ax2.set_title('Cropped profile')
    cbar2.set_clim(vmin=np.amin(thres),vmax=np.amax(thres))
    cbar2.draw_all()
    textbox.set_val('Image cropped')
    flag_save = 3

def rotate_image(event):
    global flat_profile, flag_save, unrotated

    (x1,y1), (x2,y2) = ROI
    if unrotated == 'thres':
        flat_profile = rotate(thres, [y1, y2], [x1, x2])
        im = ax1.imshow(thres)
        ax1.axis('off')
        ax1.set_title('Original/Thresholded')
        cbar1 = fig.colorbar(im, ax=ax1)
        cbar1.set_clim(vmin=np.amin(thres),vmax=np.amax(thres))
        cbar1.draw_all()
    elif unrotated == 'raw':
        flat_profile = rotate(prof, [y1, y2], [x1, x2])
        im = ax1.imshow(prof)
        ax1.axis('off')
        ax1.set_title('Original/Raw')
        cbar1 = fig.colorbar(im, ax=ax1)
        cbar1.set_clim(vmin=np.amin(prof),vmax=np.amax(prof))
        cbar1.draw_all()
    ax2.imshow(flat_profile)
    ax2.set_title('Rotated/Horizontalized')
    cbar2.set_clim(vmin=np.amin(flat_profile),vmax=np.amax(flat_profile))
    cbar2.draw_all()
    textbox.set_val('Image rotated')
    flag_save = 2

def compute_parameters(event):
    global fig2

```

```
    (x1,y1), (x2,y2) = ROI
    area = flat_profile[y1:y2,x1:x2]
#    area = thres[y1:y2,x1:x2]
#    area = prof[y1:y2,x1:x2]

    # Mean plane
    M = np.mean(area)

    # Distances of the sufrace from the mean plane
    abs_diff = np.abs(area - M)

#    # Flatness deviation
#    flt =

    # Maximum peak height of the surface
    Sp = np.amax(area-M)

    # Maximum valley depth of the surface
    Sv = np.amax(M-area)

    # Maximum height of the surface
    Sz = Sp + Sv

    # Average roughness ( Mean difference around mean plane )
    Sa = np.mean(abs_diff)

    # Roughness (Standard deviation of difference(smoothed surface, surface))
    filt = gaussian_filter(area, 11)
    diff = filt - area
    Sr = np.std(diff)

    # Root mean square roughness
    Sq = np.sqrt(np.mean(abs_diff**2))

    # Skewness (Degree of symmetry of the surface heights around the mean plane)
    Ssk = 1/Sq**3*(np.mean(abs_diff**3))

    # Kurtosis (Presence of lack of inordinately high peaks or deep valleys)
    Sku = 1/Sq**4*np.mean(abs_diff**4)

    col_labels = ['Surface\nParameter','Value','Unit','Description']
    table_data = np.array([['M','%.0f'%M,'nm','Mean plane'], ['Sp','%.0f'%Sp,'nm','Maximum peak height of the surface'],
                           ['Sv','%.0f'%Sv,'nm','Maximum valley depth of the surface'], ['Sz','%.0f'%Sz,'nm','Maximum height of the surface'],
                           ['Sa','%.0f'%Sa,'nm','Roughness: Mean difference around mean plane'],
                           ['Sr','%.0f'%Sr,'nm','Roughness: Standard deviation around surface waviness'],
                           ['Sq','%.0f'%Sq,'nm','Roughness: Root mean square difference around mean plane'],
                           ['Ssk','%.4f'%Ssk,'-','Skewness: Degree of symmetry of the surface heights around the mean plane.\n'+
                                          'Ssk>0: peaks predominance, Ssk<0: valleys predominance'],
                           ['Sku','%.4f'%Sku,'-','Kurtosis: Presence (Sku>3) of lack (Sku<3) of '+
                                          'inordinately high peaks or deep valleys']])
```

```

    # MATPLOTLIB TABLE (NOT SELECTABLE TEXT)

    fig2, ax3 =plt.subplots(1,1)
    figManager = plt.get_current_fig_manager()
    figManager.window.showMaximized()
    fig2.canvas.set_window_title('Areal Height Parameters')
    ax3.axis('tight')
    ax3.axis('off')
    tparam = ax3.table(cellText=table_data,colLabels=col_labels,
    colWidths=[0.1,0.1,0.1,0.75],
                          cellLoc='center',colLoc='center',loc='center')
    tparam.auto_set_font_size(False)
    tparam.set_fontsize(12)
    tparam.scale(1,1.6)

def average_pplot(event):
    (x1,y1), (x2,y2) = ROI
    area = flat_profile[y1:y2,x1:x2]

    ax1.cla()
    if direction == 'horiz profile':
        avg = np.average(area, axis=0)
        ax1.plot(avg)
        ax1.set_title('Average horizontal profile of ROI')
    else:
        avg = np.average(area, axis=1)
        ax1.plot(avg)
        ax1.set_title('Average vertical profile of ROI')
```

C.5 Python script: Radius of Curvature assessment (Curved profile)

This process is explained in Sec. 3.5. The **basic operations** are the following:

```
"""
Load profile image
"""
"""
...
"""

# values in um
# calibrated pixel size range
px_range = [9.8, 10.2]
px_mean = 10.0
px_uc = 0.2
sigma_z = 0.028

sel = '0'

while sel != '1' and sel != '2':
    print('NOMINAL AND MEASURED RADIUS OF CURVATURE COMPARISON\n\n')
    print('Enter \'1\' if nominal ROC is known...\n')
    print('Enter \'2\' if refractive index and focal distance of lens
    are known...\n')

    sel = input()

if sel == '1':
    ROC = float(input('Enter ROC (mm)\n'))
```

```
    f = float(input('Enter nominal focal distance (mm)\n'))
elif sel == '2':
    n_lens = float(input('Enter nominal refractive index of the lens...\n'))
    f = float(input('Enter nominal focal distance (mm)\n'))
    ROC = abs(f*(n_lens-1.0))

"""
BEST-FIT CALCULATION OF 1) 2D PROFILE WITH A PARABOLOID
              2) 1D PROFILE PASSING THROUGH CENTER WITH A PARABOLA
FOR UPPER AND LOWER LIMIT OF PIXEL SIZE
"""

for i in range(2):
    px = px_range[i]

    fringearea_gf = gaussian_filter(fringearea, sigma=1)

    if f < 0:
        [r0], [c0] = np.where(fringearea_gf == np.amin(fringearea_gf))
        Z = fringearea - fringearea[r0,c0]
        radius_c = ROC*1000
    elif f > 0:
        [r0], [c0] = np.where(fringearea_gf == np.amax(fringearea_gf))
        Z = fringearea - fringearea[r0,c0]
        radius_c = - ROC*1000


    X = np.arange(c)*px
    Y = np.arange(r)*px
    X, Y = np.meshgrid(X, Y)

    guess = [radius_c, px*r0, px*c0, 0.0]

    Z = Z/1000 # now in um

    po, pc = curve_fit(paraboloid, (X,Y), Z.ravel(), p0=guess)
    paraboloid_fit = paraboloid((X,Y), *po).reshape(np.shape(Z))
    perr = np.sqrt(np.diag(pc))

    po_list.append(po)
    pc_list.append(pc)
    perr_list.append(perr)

    if f < 0:
        [r0], [c0] = np.where(fringearea_gf == np.amin(fringearea_gf))
        z = fringearea[r0]-np.amin(fringearea[r0])
        radius_c = ROC*1000
    elif f > 0:
        [r0], [c0] = np.where(fringearea_gf == np.amax(fringearea_gf))
        z = fringearea[r0]-np.amax(fringearea[r0])
        radius_c = - ROC*1000

    guess2 = [radius_c, px*c0]

    z = z/1000 # now in um

    x = np.arange(c)*px
    s_z = sigma_z*np.ones(c)
    po2, pc2 = curve_fit(parabola, x, z, p0=guess2, sigma=s_z , absolute_sigma=False)
    perr2 = np.sqrt(np.diag(pc2))

```

```
        po2_list.append(po2)
        pc2_list.append(pc2)
        perr2_list.append(perr2)

    """
    STATISTICS
    """

        residuals = Z.ravel() - paraboloid((X,Y), *po)
        ss_res = np.sum(residuals**2)
        ss_tot = np.sum((Z.ravel()-np.mean(Z.ravel()))**2)
        r_squared = 1 - (ss_res / ss_tot)

        r_squared_list.append(r_squared)

"""
RESULTS: ROC AND UNCERTAINTY IN MEASUREMENT
"""
# values in mm
error_pxmin = perr_list[0][0]/1000
error_pxmax = perr_list[1][0]/1000
ROCmeas_pxmin = abs(po_list[0][0])/1000
ROCmeas_pxmax = abs(po_list[1][0])/1000
minROC = ROCmeas_pxmin - error_pxmin
maxROC = ROCmeas_pxmax + error_pxmax
ROC_meas = (maxROC + minROC)/2
ROC_error = (maxROC - minROC)/2
```

Appendix D

Integrated Graphical User Interface (GUI)

For the shake of operational simplicity of the whole procedure from the interferogram capturing to the profile analysis, a minimalistic GUI was developed (Fig:D.1, D.2). The requirements of the preparation for measurement are such that many procedures has to be done in parallel. For this reason, the code of this GUI was built strongly based on the "multiprocessing" package of Python. To run the GUI, simply run the "WLtopo_mp.py" script from the Windows command prompt (some IDEs may be problematic with running multi-processing modules in full functionality).

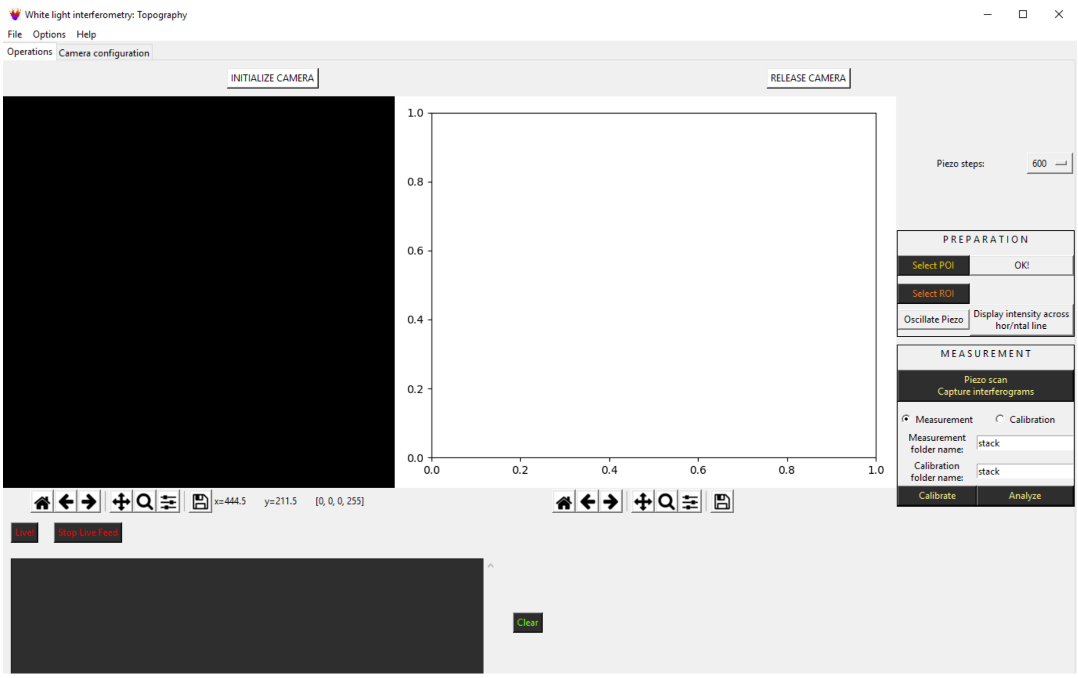

FIGURE D.1: GUI main window: operations and graph planes for results.

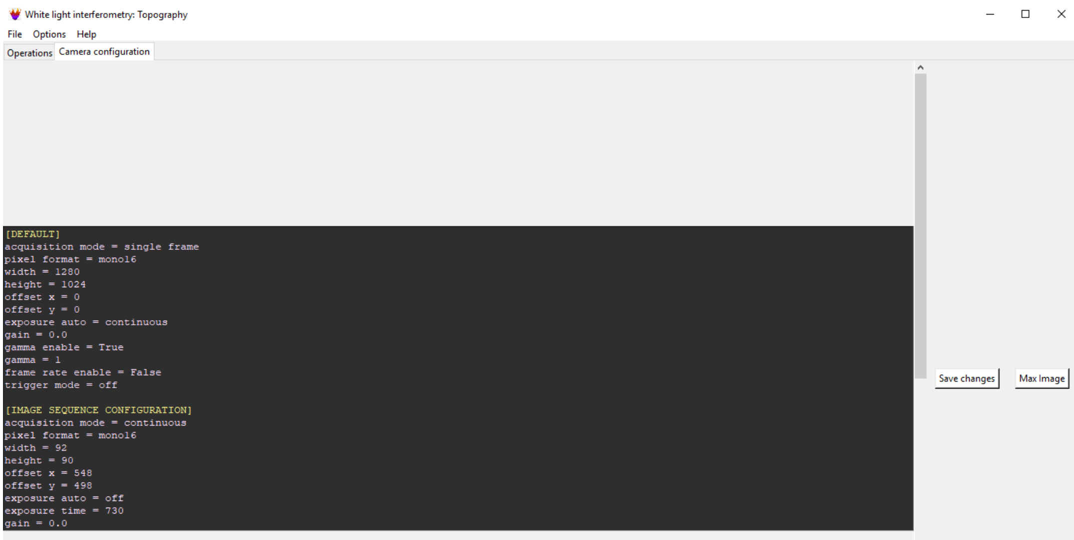

FIGURE D.2: GUI camera configuration window: all basic options needed for the camera to run.

–FUNCTIONS–

CAMERA CONFIGURATION TAB

Before taking a measurement, many things must be done in order, so as to achieve the best possible measurement. For starters, the user connects the camera, initializes it and configures it according to the "config.ini" file. A copy of it is stored in an editable text box in the "camera confiration tab". The user can change every single camera option of each section. There are three sections: "*Default*", which uses the maximum possible image size, "*Image Sequence Configuration*", which uses 16-bit pixel depth and internal software trigger for capturing each image and "*Live View Configuration*", which does not use any trigger but has 8-bit pixel depth for displaying purposes. Any other camera option is irrelevant to the occasion used and freely changes by just overwriting and saving the corresponding values.

LIVE VIEW

On the left of the "Operations" tab, the "Live View" window displays continuously the selected region of interest of the sample. Press "Live!" and "Stop Live feed" to start and stop respectively the live feed.

PREPERATION FRAME

Select ROI

All operations here have the purpose of setting up the sample holder of the interferometer in the best possible position in the x,y,z axes, with the most important being the axis parallel to the optical one.

- Click on the "Select ROI" button.
- Click on the "Live View" WIndow on the wanted top left corner of the ROI and release on the bottom right one. A box selection has been made.
- Modify the selection by clicking and releasing around the box.
- When ready, press enter.

Display intensity across horizontal line

In some cases when seeking interference fringes, it might be useful to have an auxiliary live plot of just a line across the image to search for deviations in the constant light intensity that may indicate the existence of fringes. In that case:

- Press "Display intensity across horizontal line". Light intensity vs horizontal image axis plot is displayed on the right window/plane in synchronization with the live view feed on the left window.
- Press the same button to stop the plotting.

Oscillate Piezo

WLSI method requires the existence of a well shaped interference pattern per sample point (signal per pixel) from which the argmax will be easily determined, inferring thus the height of this sample point. So to avoid marginal situations where the argmax is located very close to the beginning or the end of the "per pixel signal", an auxiliary function can be used.

The user can select multiple Points of Interest (POI):

- Click on the "Select POI" button.
- Click on the "Live View" window on the wanted POI. Press enter.
- If more points of interest are needed, select each one and press enter.
- When every POI is selected, to finish the process click on the "OK" button next to the "Select POI" button.
- Press "Live!" to start the live feed.
- Press "Oscillate Piezo" to start the back and forth oscillation of the piezo. In each movement of the piezo,

MEASUREMENT FRAME

Piezo scan – Capture interferograms

Following preperation, for the WLSI measurement and calibration of the piezo movements, a piezo scanning must be done for each case:

- Type a measurement/calibration folder name (which will contain the interferograms) on the corresponding text field.
- Select "Measurement" or "Calibration" from the radio buttons.
- Click on "Piezo scan – Capture interferograms" button to execute the image acquisition. [Note: The calibration process requires the proper use of the optical arrangement (Coherent light source, optical flat/mirror in reference arm)]

Calibrate

Before the first measurement, a calibration must be done regarding the piezo movements, that is the steps and real distance made correspondence. [Note: The calibration process requires the proper use of the optical arrangement (Coherent light source, optical flat/mirror in sample arm)]. After the aqcuisition of the images as described above, the user will:

- Click on the "Calibrate" button. When the process is done, a text file containing the piezo displacements between each scan will be saved on the root folder with name "Mapping_Steps_Displacement_<piezo steps>_<file increment number>.txt"

Analyze

The final step to the whole process for obtaining a 2D profile of a specimen is the analysis of the measurement interferograms:

- Click on the "Analyze" button.
- Select the interferograms image set from the emerging prompt window.
- When the profile is calculated, it will be saved on the root folder with name "raw_profile_<interferograms image set folder name>" in text and tiff format.
- A new window will emerge for *post processing* of the profile. These functions are *image cropping*, *histogram thresholding*, *average line profiling* and *rotation of the profile plane parallel to the ground.*

–PARALLELIZATION–

The workload specifically for live continuous image display and synchronized feature plotting is beyond the capabilities of usual single-threaded script/modules. In this way, each new operation starts only when the previous in order has finished. To overcome this, a multi-processing/multi-threading approach was implemented which allows the parallel execution of at least two different operations.

Two extra processes are created when running the GUI script "WLtopo_mp.py". More specifically, there exist a process for *GUI design and display* which actually creates a child process for *camera manipulation* purposes and another for *piezo oscillation* purposes. In order to have synchronized camera displaying, piezo movement and feature plotting, these three processes must communicate with each other. Every time some data are needed to be plotted either on "Live View" window or in the adjecent plane (feature plotting), the camera manipulation process transfer these data to a shared queue. Then, the GUI process reads the data from the queue and displays/plots them accordingly. The whole operation happens in a datum per datum manner to keep a synchronized data "production" and "consuption".