LIDROTEC

Laser Scanning Microscope Macro with python

Background

Objective of the project is to automatically generate, evaluate and compose large data points (5000-10000) generated by the laser process with help of the Laser-Scanning Microscope. Later use machine learning to optimize the laser process within short period of time.

This includes

- Generating patterns for the laser process.
- Automatically measure the patterns to generate data with the Laser-Scanning Microscope.
- Automatically evaluate the data points regarding the process outcomes.
- Correlate the process outcome with the process parameters – first manually – later with machine learning.

This document explains python program that automatically measure the patterns to generate data with the Laser-Scanning Microscope.

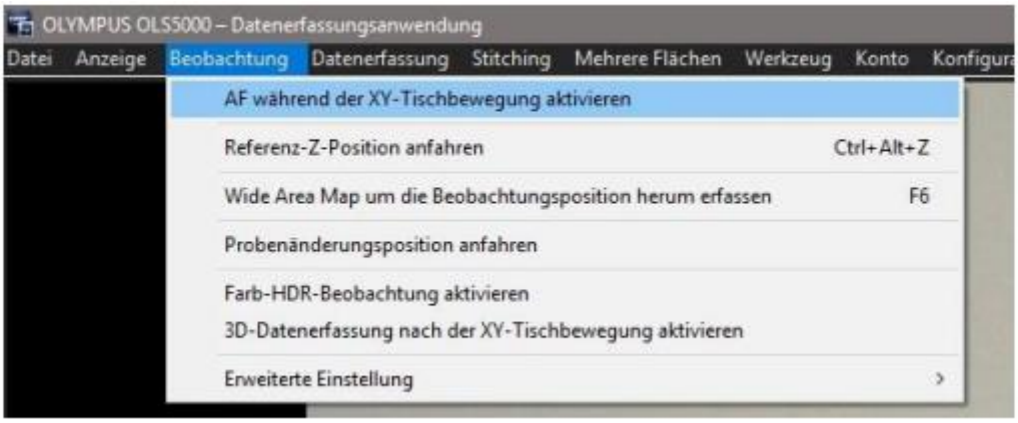
Procedure

A class was created in python to help with the creation of a macro. In this class, the programme frame is created and individual functions are combined into a sequence. As the distances between the measurement fields are known, they can be recorded and evaluated with the generated report from Laser scanning microscope.

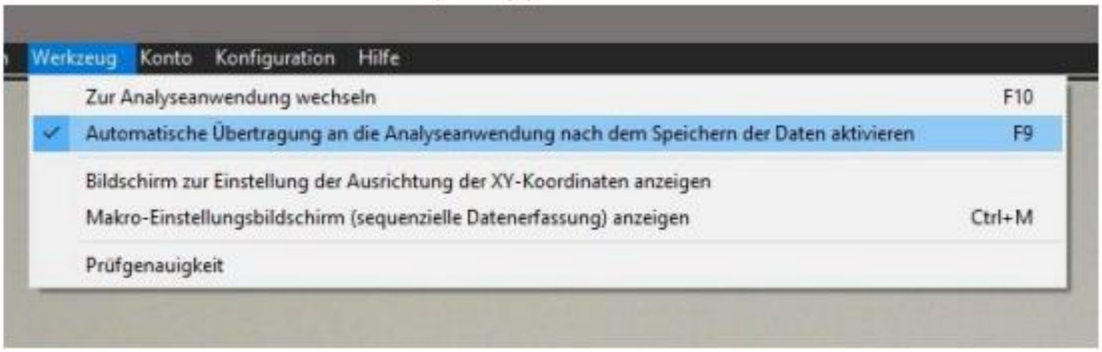
Necessary settings

These are the settings necessary for the macro to work in Laser scanning microscope software.

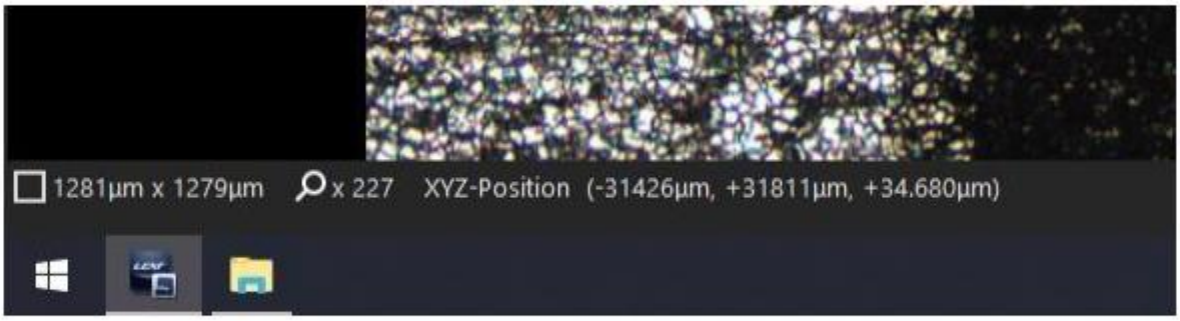
1. Autofocus during XY table movement should be deactivated.


OLYMPUS OLS5000 – Datenerfassungsanwendung
Datei
Anzeige
Beobachtung
Datenerfassung
Stitching
Mehrere Flächen
Werkzeug
Konto
Konfigura
AF während der XY-Tischbewegung aktivieren
Referenz-Z-Position anfahren
Ctrl+Alt+Z
Wide Area Map um die Beobachtungsposition herum erfassen
F6
Probenänderungsposition anfahren
Farb-HDR-Beobachtung aktivieren
3D-Datenerfassung nach der XY-Tischbewegung aktivieren
Erweiterte Einstellung

2. Automatic transmission to the analysis application must be activated.


Werkzeug
Konto
Konfiguration
Hilfe
Zur Analyseanwendung wechseln F10
✓ Automatische Übertragung an die Analyseanwendung nach dem Speichern der Daten aktivieren F9
Bildschirm zur Einstellung der Ausrichtung der XY-Koordinaten anzeigen
Makro-Einstellungsbildschirm (sequenzielle Datenerfassung) anzeigen Ctrl+M
Prüfgenauigkeit

3. The starting position or a reference position of the test field must be known.



Class and methods

The following section describes about class created in python to help with the creation of a macro and the methods used in the programme.

Class	LSMMacroClass
Function	Generates the macro class object. All functions are called with instance of this class object.
Call	LSMMC = LSMMakroClass()
attributes	macroFileName : string variable of the macro file name. Example, LSMMC.macroFileName = 'xyz.mcr'
comments	• LSMMC was chosen arbitrarily here and, like any variable name, can be any combination of letters and numbers, although care must be taken that variables never begin with a number. • The following functions in this document are called with a dot between the handle and the variable . Example: LSMMC.generateFile()

Command	generateFile
Function	Generates the macro and writes the necessary commands into the macro file.
Call	LSMMC.generateFile()
Parameters	none
comments	• This method generate macro file with name variable assigned to ‘macroFileName’.

Following methods are used privately in macro class and are not required while creating macro class object:

Command	importDummyMacro
Function	Imports the dummy macro that contains all macro methods to a new file with line numbering that starts from zero.
Call	importDummyMacro()
Parameters	none
comments	• This function is important to copy required macro functions to the new macro file using line numbering. • There is no need to call the method when instantiating as it is called within another method generateFile().

Command	copyDummyLines
Function	Copy the required macro command lines from the imported file by importDummyMacro.
Call	copyDummyLines(nStart, nEnd)
Parameters	nStart: integer variable indicating start of the line. nEnd: integer variable indicating the end of the line.
comments	

Command	writeLine
Function	add the required modification to macro commands copied from the .bak file.
Call	writeLine(text)
Parameters	text: string variable
comments	

Methods used while creating macro file using LSMMacroClass():

Command	comSetAutoSaveSetting
Function	Sets the folder and the file name on the laser scanning microscope for automatic saving of data to the assigned directory.
Call	LSMMC.comSetAutoSaveSetting(folderName, fileName)
Parameters	folderName: string variable, denotes the path to directory to save the reports generated by the laser scanning microscope. fileName: string variable, file name to save the generated LSM reports.
comments	• Example: LSMMC.comSetAutoSaveSetting('D:\Data\Test', 'Test01')

Command	comSetReportTemplate
Function	Stores the template for the evaluation of the Data (e.g. determine depth and roughness)
Call	LSMMC.comSetReportTemplate(fileName)
Parameters	fileName: string variable, file name to save the generated LSM reports.
comments	• Method argument should include the entire directory path. Example: LSMMC.comSetReportTemplate('D:\Data\Test\test01.tpl')

Command	comSetAlignment
Function	Stores the alignment template for the transforming to new coordinate according to the specimen.
Call	comSetAlignment(fileName)
Parameters	fileName: string variable, file name to set as alignment template.
comments	• Method argument should include the entire directory path. Example: LSMMC.comSetAlignment('D:\Data\Test\alignment.isc')

Command	comMoveXYZStage
Function	Move to the position xPos and yPos.
Call	LSMMC. comMoveXYZStage(xPos, yPos)
Parameters	xPos,yPos: float variables
comments	• The specificied measurements in the arguments are in micrometers.

Command	comSnapshot
Function	Creates a snapshot at this current position.
Call	LSMMC.comSnapshot()
Parameters	fileName: string variable, file name to save the generated LSM reports.
comments	• Method can be used to check the measurement of a specific location is carried out. • This command is necessary if the table is moved and stitching is to take place at the current location.

Command	comSetMultiPointArea
Function	Stitches the multiple area together and measure it as a one big area.
Call	LSMMC. comSetMultiPointArea(nRow, nCol)
Parameters	nRow, nCol: integer variable denotes the number of rows and column of area required to merge it into one measurement location.
comments	• it is only possible to stitch from the top left to the bottom right.

Command	com3DExtendedAcquisition
Function	Starts the 3D data acquisition.
Call	LSMMC.com3DExtendedAcquisition()

Parameters	none
comments	• This is set to automatic in the current version. It must therefore be ensured that the sample does not have any jumps. • The measurement is completed automatically.

Command	comSaveReport
Function	After the measurement, the application automatically switches to save report with this method.
Call	LSMMC.comSaveReport(N, ext)
Parameters	N: integer variable, specified number to save the report. ext: string variable, file extension to save the laser scanning microscope generated report.
comments	• The number is stored in 4-digit format with preceding numbers, Example: 0004. • The file extension types currently available are ‘EXCEL’, 'PDF' and 'REP'. Example: LSMMC. comSaveReport(4, 'PDF')

Command	comChangeRevolver
Function	Objective of the laser scanning microscope can be changed.
Call	LSMMC.comChangeRevolver(N)
Parameters	N: integer variable, corresponding to the required objective.
comments	• Each of the objective are assigned a value from 1 to 5 in the laser scanning microscope. • For the safety reason N can be only values from 1 to 3 (only for low power objective).

Command	comCloseReport()
Function	After saving is complete, the report can be closed.
Call	LSMMC.comCloseReport()
Parameters	none
comments	• This is useful when many reports are open.

Command	CloseFile()
Function	closes the file properly
Call	LSMMC.closeFile()
Parameters	none

Laser scanning microscope macro demo

```
# LSMMacroDemo
# Generate the makro file
# In this tutorial you will learn the basics of yout macro.
# The following commands are fundamental to generate your macro.

from LSMMacroClass import LSMMacroClass # importing class from package

if __name__ == "__main__":

          # name of the generated macrofile using LSMMacroClass
          newMacroFileName = 'python_macro_test1.mcr'

          # open the macro class and send it to a handle
          LSMMC = LSMMacroClass()

          # tell the handle the macro file name
          LSMMC.macroFileName = newMacroFileName

          # now generate the basics of the makro file
          LSMMC.generateFile()

          # start by setting the PATH and NAME for auto saving the data
          LSMMC.comSetAutoSaveSetting(r'D:\Data\Aswin\Automation\python_macro',
'python_macro_test')

          # *** Define your template prior to your measurements! Note that it is
         # important to tell the Acquisition program that it has to analyze measured
         # data automatically, see the instruction for more details.***
          LSMMC.comSetReportTemplate(r'D:\Data\Aswin\Automation\python_macro\test.tpl')

          # define your alignment to set the coordinates for measurement.
          LSMMC.comSetAlignment(r'D:\Data\Aswin\Automation\python_macro\alignment.isc')

          # *** You need to know your starting point -
         # Define your x, y coordinates for the starting point or make it zero if you have
         # set alignment template. This may be the first area
         # of your stitching (normally top left).
         # The coordinates are given in MICROMETERS! ***
          xStart = 1000 # µm
          yStart = -1000 # µm

          # *** Here the variable for the report number is set to zero, because it
           # will be increased later. ***
          repNr = 0

           # *** The following for-loop makes it possible to take measurements at the
           # starting point and in 3 millimeter steps until to 6 millimeters from the
           # starting x-point and y axis. Note that the
           # positions are given in MICROMETERES! ***
          for iy in range(0,6000,3000):
                    for ix in range(0,6000,3000):
                              # Tell the LSM that it has to move to another position.
                              LSMMC.comMoveXYZStage(xStart+ix,yStart-iy)

                              LSMMC.comSnapshot(1)

                              # *** set the stitching area - here 2 by 2 (x / y) fields are
measured. Keep in mind that the size of the field depends on your objective. ***
                              LSMMC.comSetMultiPointArea(1,1)

                              # now start your measurements
                              LSMMC.com3DExtendedAcquisition()

                              # save report as PDF-file
```

```
                             LSMMC.comSaveReport(repNr,'PDF')

                             # *** After you have saved your report, you should close the
report. If you
            # don't close it, and you analyze hundred of test fields, you may imagine
            # what will happen to your computer... ***
                             LSMMC.comCloseReport()
                             repNr = repNr + 1

          # changes the objective 1 to 2
          LSMMC.comChangeRevolver(2)

          LSMMC.closeFile()
```

Macro errors

This section mentions possible macro error and its solution:

- “Failed in executing command”
  ➔ Check whether all given directories of the measurement templates are correct.
- “Connection error has occurred, Please check the connection”
  ➔ Make sure that analysis application is running.
- “Failed in executing command”
  ➔ Enable “start analysis application after saving the data” from tools in laser scanning microscope.

Creating a macro command

This section briefly describes the creation of new macro methods in the macro class "LSMMacroClass.py".Here the creation of method "comSetAutoSaveSetting" is explained. This is intended to add new commands that are not yet available in the class.

- The function call in the programme "MacroClassTest.py" is comSetAutoSaveSetting(folderName, fileName).
- folderName and fileName are the arguments required for the method.
- The variables folderName and fileName are initially stored in class variables. This way they are also available to other functions if necessary. (obj.ASFolderName = folderName, obj.ASFileName = fileName)
- Next, lines 5 to 10 are copied from the dummyMakro.bak into the macro file. These are commands or macro information that are not influenced by the user (obj.copyDummyLines(4,10))
- Then the information of the file name (fileName) is stored in the macro. (obj.writeLine([' ' fileName '']);)
- Lines 12 to 17 are copied before the folder name (folderName) is also stored in the macro document.
- Finally, lines 19 to 72 must be copied in order to correctly store the command for the automatic saving settings in the macro.

  This example can be used to illustrate why the file "dummyMakro.bak" is important. For the settings of the automatic saving, 68 lines are necessary in the macro. However, only 2 of the 68 lines are changed. These are the lines with the file and folder names. In order to keep the functions in the class clear, all the information for the commands that cannot be changed is copied from the file "dummyMakro.bak". Only the two lines with the changed values are rewritten.

Conclusion

The macro generated with this python class consist of many sub-commands and variables that can be changed. In this document, only the necessary parameters have been inserted (e.g. file names and file location). If other parameters need to be changed, this can be done by modifying this macro class. The file dummyMakro.bak (read-only) should be saved in the same folder as the class file in order to successfully execute the macro class.