Guided Acquisition

Advanced Automated Microscopy

Carl Zeiss Microscopy GmbH
Carl-Zeiss-Promenade 10
07745 Jena
Germany

microscopy@zeiss.com
www.zeiss.com/microscopy

Carl Zeiss Microscopy GmbH
ZEISS Group
Kistlerhofstr. 75 81379 München
Germany

Effective from: 01 / 2018

Contents

1 Introduction

For a growing number of applications, it will be crucial to acquire data in a smart way. One way to achieve this goal is to build a smart microscope, which essentially means creating smart software workflows to control the hardware based on image analysis results.

Idea or Task:

- Scan or inspect a large area.
- Detect an "interesting" object.
- Acquire detailed data for every event.
- Automate the workflow to minimize user interference.

First of all it is important to define what a Object of Interest can actually be. Example would be an object that meets specific criteria, for example:

- size
- brightness
- shape
- intensity
- combinations of the above

It could be something quite simple. For instance one can have lots of cells, that are stained with blue dye, and only a few of them (maybe where the transfection worked . . .) are also expressing GFP. The idea here would be to detect all cells that are positive for both colors and acquire an z-Stack for every cell (blue & green) that meets those criteria. Therefore this kind of application requires three major tasks:

1. Define the Overview Scan Experiment.
2. Define the object detection rules, e.g. setup image analysis.
3. Define the Detailed Scan(s) to be carried out in case of a "positive" object.

Depending on the application the image analysis step can be done by using

- ZEN Image Analaysis incl.Machine-Learning methods (Guided Acquisition module)
- Running an APEER module inside a docker container (script version)
- Calling an external application (script version)

INFO:

The option Running an APEER module required to have Docker installed on the system, an APEER account and obviously a module that created a table containing the correct stage coordinated for the object of interest. For more info on APEER please visit: http://www.apeer.com

1.1 Basic Workflow Definition

The main workflow can be summarized as follows:

1. Acquire some sample data showing a object of interest.
2. Setup an Image Analysis Pipeline to detect those events or ...
3. ... create an APEER module
4. Define an experiment which does the Overview Scan.
5. Define an experiment which does the Detailed Scan.
6. Automate the entire workflow.

The goal of this tutorial is to create an automated workflow that can be used to easily setup a Guided Acquisition. This requires some knowledge about the OAD macro environment and its scripting language Python.

1.2 General Workflow Definition

This is the outline of the general workflow, which would be automated.

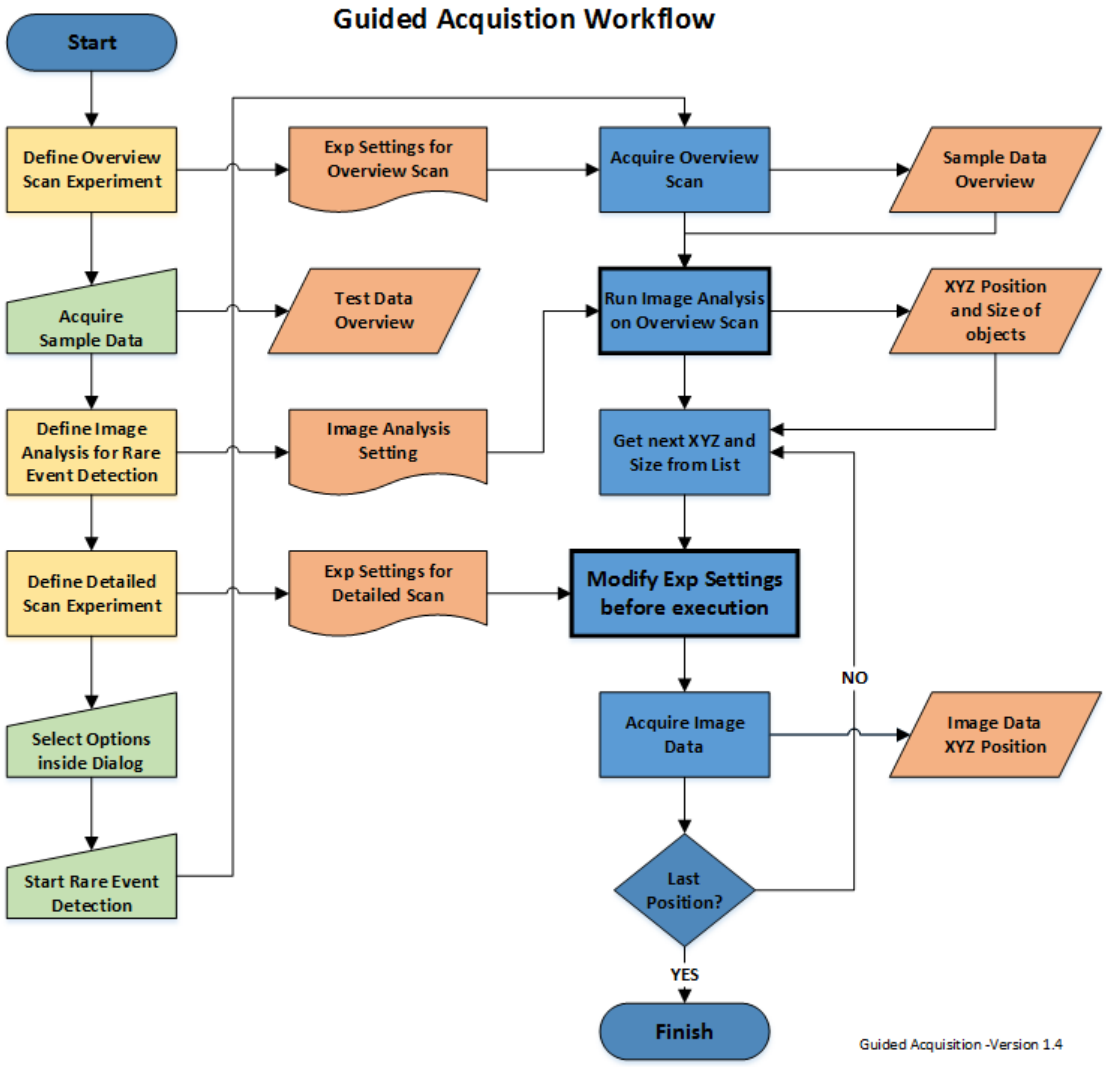

Figure 1: General workflow for Guided Acquisition.

A different way to visualize this is shown in the figure below.

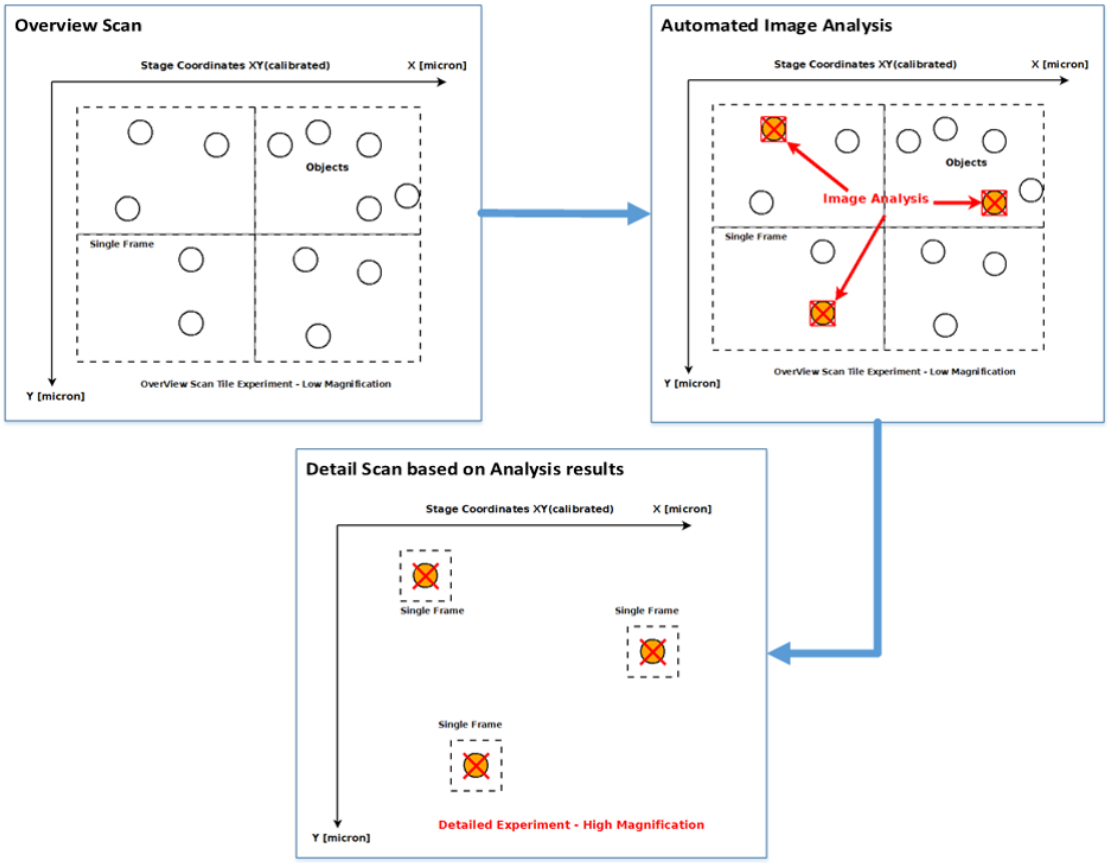

Figure 2: Larger area with some candidates for a "interesting" objects.

1.3 General Workflow - Complete Overview

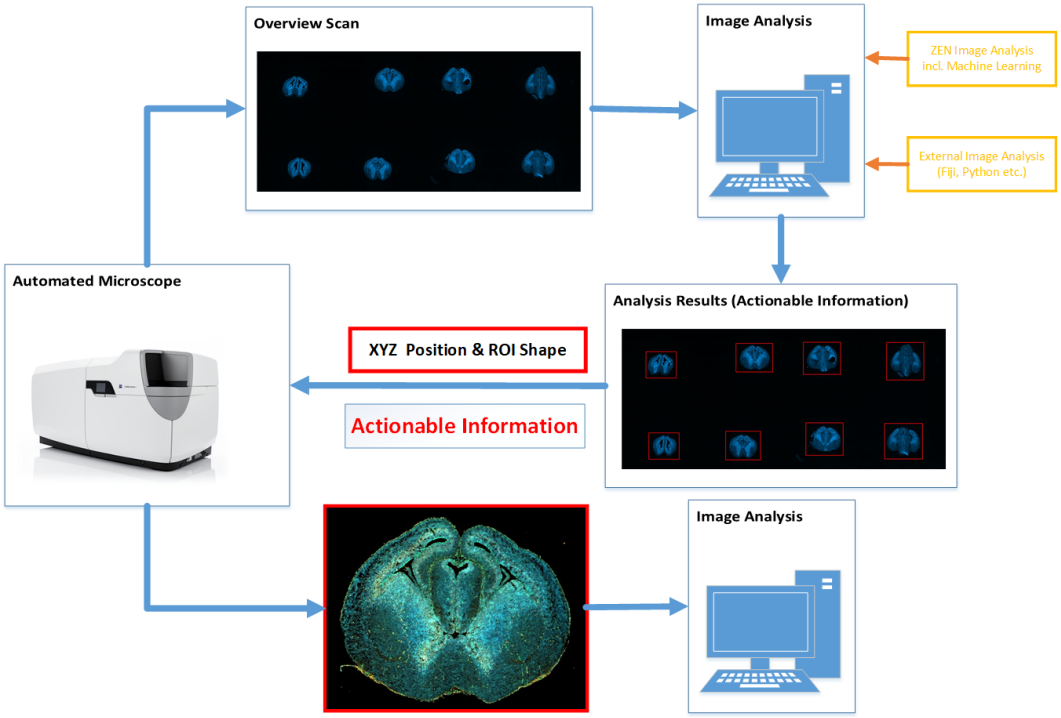

Figure 3: Workflow - Actionable Information 1.

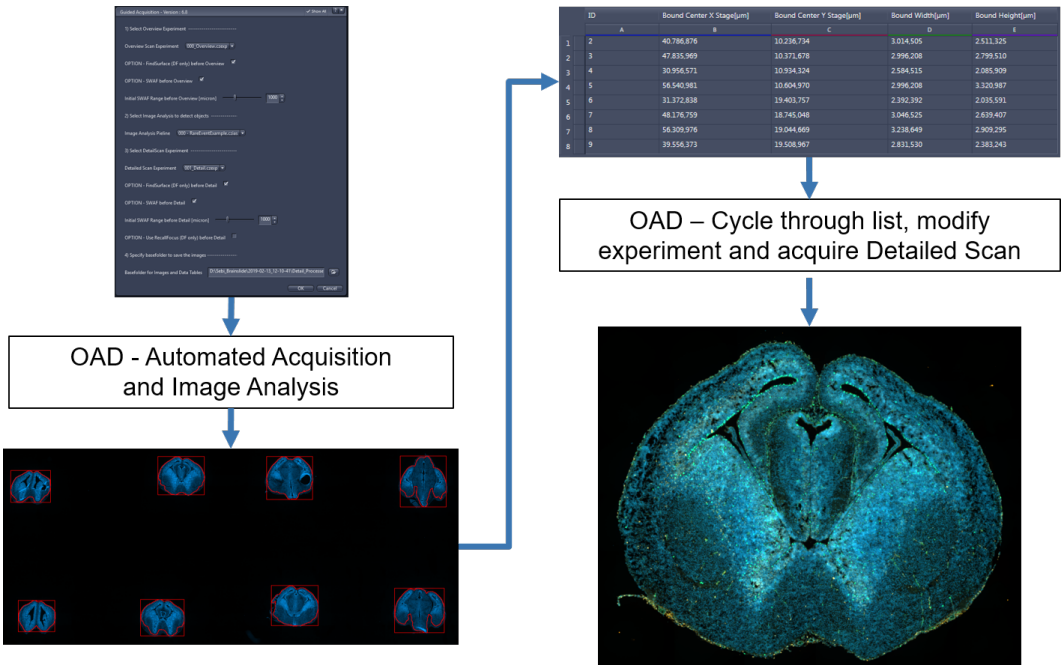

Figure 4: Workflow - Actionable Information 2

2 Workflow - Single Steps

This part of the tutorial will explain all required steps to set up the complete workflow in more detail. Some knowledge about image analysis is required here.

2.1 ZEN Prerequisites

For automated workflows like Guided Acquisition is it mandatory to switch off any modal windows, that would require the user to press a button during the preparation of an experiment run. An example is the reminder message box during the initialization of the Definte Focus.

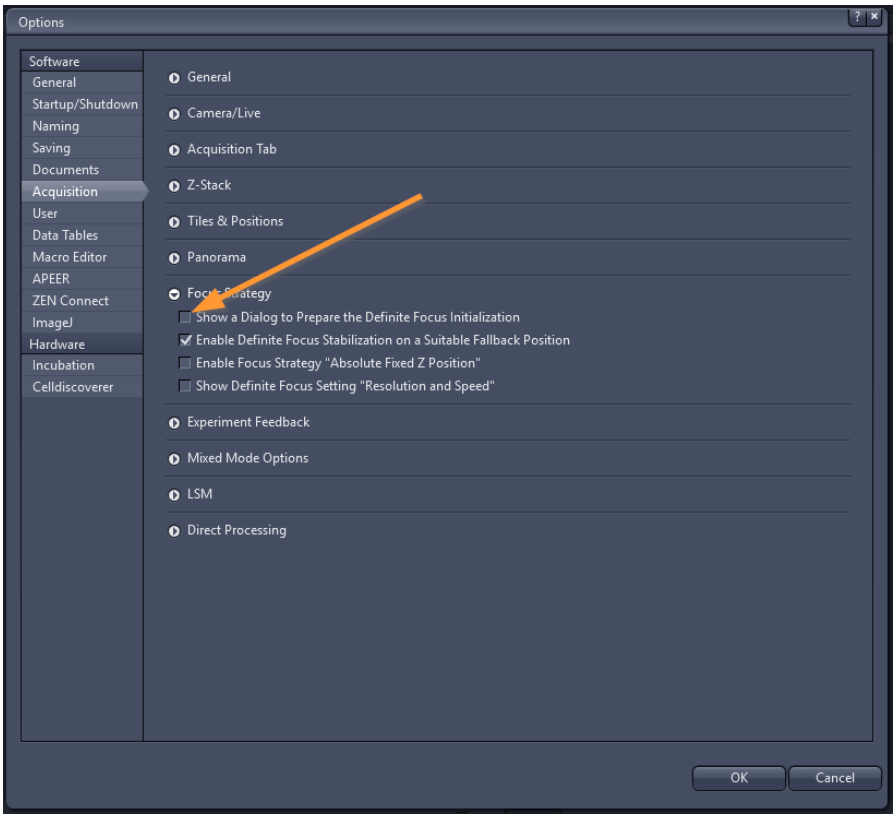

Figure 5: Prerequisites - Disable Message Box

INFO:

The option Show a Dialog to Prepare the Definite Focus Initialization should be disabled. The only exception would using Focus Strategies using multiple z-Offsets that require user interaction in setting them up.

2.2 Acquire Sample Data with Overview Scan

The first crucial step is to have a basic idea of what the actual rare event will look like, inside an real image acquired, with the appropriate parameters (light intensity, detector settings, filters, objective, ...). An ideal sample data set should be acquired using the "real" acquisition parameters will be used to define the overview scan experiment later on.
Typically such an overview scan is acquired using a lower magnification in combination with a tile experiment. The exact parameters will be specific for the application, but the main idea is always the same:

The overview image must contain the information to locate the objects of interest based on "some" features that can be retrieved via an appropriate image analysis.

Once a representative sample data set is available, one can start setting up an image analysis pipeline. ZEN offers currently two ways do to so:

1. The easy way – Use the ZEN Analysis Wizard.
2. Program your own image analysis pipeline using an OAD macro (this is not covered by this tutorial).
3. Use an APEER module to detetct the objects of interest and create the required table
4. Program your own image analysis using an external software (e.g. Fiji, Python etc.) and use it from within ZEN (this is not covered by this tutorial).

The most comfortable way is to use the wizard. this offers enough flexibility to cover most of the applications and allows to use trained machine-learning models to segment the image as well.

INFO:

AN example for an APEER module suitable for the Guided Acquisition workflow can be found here: https://github.com/zeiss-microscopy/OAD/tree/master/Apeer/modules/segmentobjects.

2.3 Create Image Analysis Pipeline

Here one already sees a "half-ready" image analysis pipeline. In order to automate, the wizard step containing the feature selection is crucial.

In order to relocate all detected objects, it is of course required to retrieve the current XY(Z) position of every detected event.

The main idea is to determine all parameters required to get access to the stage coordinates of a detected object.

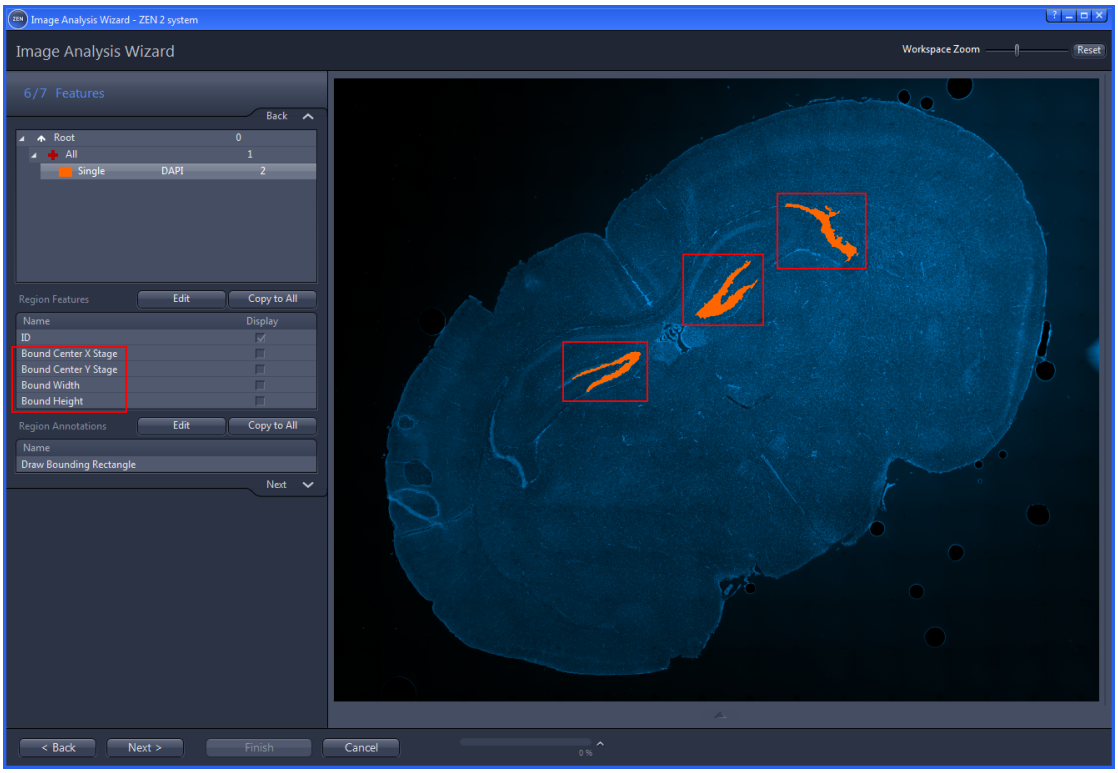

Figure 6: Select the required positional features inside the wizard.

Currently one needs to select the following parameters when creating rectangular TileRegions bases on the Image Anaylsis:

- Object ID
- Bound Center Stage X
- Bound Center Stage Y
- Bound Width
- Bound Height
- ImageSceneContainerName

The object ID is needed to keep track of the executed detail scans and to save the resulting image files with the correct ID as part of the used filename.

The parameters Bound Center Stage X and Bound Stage Center Y yield absolute stage coordinates of the detected objects. These will be used to relocate the objects for the detailed scan later on.

The parameters Bound Width and Bound Height yield in the absolute width and height of the bounding rectangle in [micron]. This is required if the detected object is bigger than a single frame. In such a case the tile region of the detailed experiment will be adapted to the size of the bounding rectangle automatically.

The parameter ImageSceneContainerName is needed to keep track of the respective wells, because the wellID will be used as part of the filename for the detail scan. Depending on your application, it might be useful to measure additional parameters. Feel free to add whatever might be useful.

INFO:

The parameter ImageSceneContainerName is not required under all circumstances and only makes sense if there really is a physical well or container on your sample.

When choosing the option to create Polygon TileRegions BoundWidth and Bound Height are not required but it is recommended to include them anyway

2.4 Define Overview Experiment

Usually this is done using the Tiles and Positions module to scan a large area. This tutorial assumes that one is already familiar with this ZEN module.

Important Remark: Since one wants to relocate the detected objects, it is required to calibrate the XY stage before the overview scan, and a rigid and stiff sample holder should be used.

For this tutorial one can use an already acquired image representing the real overview scan (instead of a real experiment). The shown image is a 16x13 tile image acquired with a 10X magnification. The result of such a scan followed by the image analysis is shown here:

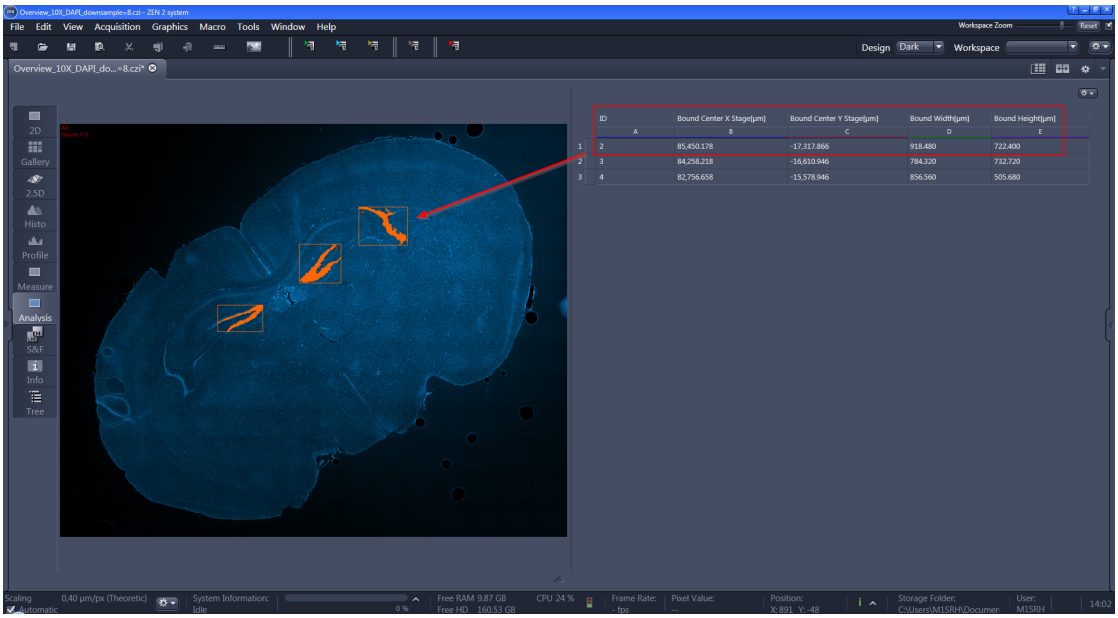

Figure 7: Result of the image analysis on the overview scan image.

2.5 Define Detailed Scan Experiment

First of all it is important to understand, that there is no such thing as a typical "Detailed Scan". What this experiment (or even a workflow) might be, depends on the application. In a general sense such a detailed scan is "some kind" of experiment carried at a specific position based upon the results of the image analysis. Examples could be:

- Simple Z-Stack with a high-NA objective lens.
- Multi-Channel Z-Stack using an optical sectioning method like SD, Apotome or AiryScan
- One of the above combined with a tile experiment.
- A series of subsequent experiments with different image modalities.

Since this concept is based on OAD, it is possible to automated almost any kind of workflow at a given position.

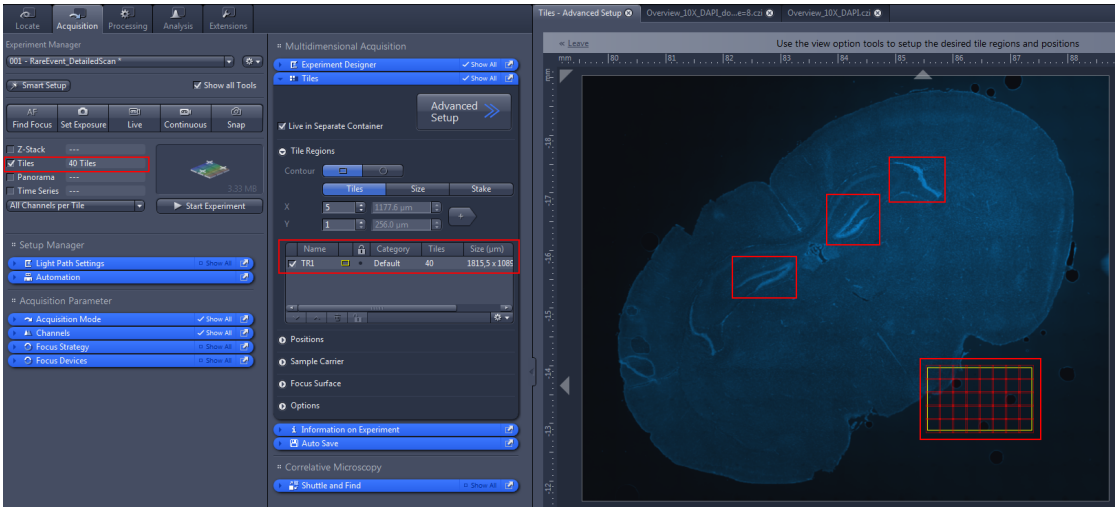

Figure 8: Setup of a simple detailed scan experiment.

Now the preparation is complete. So far we have available:

- The image analysis pipeline based on some sample data
- The Overview Scan Experiment = tile experiment created by the Tiles & Positions Module
- The Detailed Scan Experiment, which is also a tile experiment.

And now the real interesting part begins – how to setup a complete workflow out of those building blocks using an OAD Python script inside ZEN Blue.

3 Automation via OAD

3.1 Prerequisites

In order to be able to run the script one needs to copy it into the correct folder, which is located at:

"c:/Users/XXXX/Documents/Carl Zeiss/ZEN/Documents/Macros"

where XXXX stand for the actual user name.

3.2 Pure Scripting or User Dialog

First of all one has to decide if a pure script is sufficient or if there should be some kind of simple user interface (very basic wizard).

- A pure script is of course the faster solution, but is may be difficult to use for less advanced users.
- A basic wizard offers less flexibility, but may be easier to use for most users.

The choice clearly depends on the user, since there is no right or wrong approach here. For this tutorial we will focus on the 2nd approach. Once one has figured out how to realize this, the 1st approach will be straight forward since the basic workflow is identical.

3.3 User Dialog Requirements

The user interface of the dialog will following functionality:

1. Select the overview scan experiment.
2. Option to use Definite Focus (DF2) before overview scan
3. Option to use SWAF before Overview
4. Define range for SWAF
5. Define ZEN Image Analysis Settings to be used
6. Option to Polygon TileRegions instead of BoundingBox
7. Option to use APEER module instead of ZEN Image Analysis
8. Define the APEER module setting to be used
9. Select the detail scan experiment

10. Option to use Definite Focus before overview scan
11. Option to use SWAF before overview scan
12. Define range for SWAF
13. Option to use Recall Focus using (DF2)before detail scan
14. Select a folder to store the image data and analysis results

3.4 Create the User Dialog

The relevant part of the script is shown below. Once the dialog is closed, the results have to be collected.
Additionally one must check if the Detailed Scan experiment is a tile experiment, where the tile size has to be adapted accordingly. This is done using a little tool function.

```
538 # Initialize Dialog
539 GuidedAcqDialog = ZenWindow()
540 GuidedAcqDialog.Initialize('Guided Acquisition - Version : ' + str(version))
541 # add components to dialog
542 GuidedAcqDialog.AddLabel('------   Select Overview Experiment   ------')
543 GuidedAcqDialog.AddDropDown('overview_exp', 'Overview Scan Experiment', expfiles_short, 0)
544 GuidedAcqDialog.AddCheckbox('fs_before_overview', 'OPTION - FindSurface (DF only) before Overview', False)
545 GuidedAcqDialog.AddCheckbox('SWAF_before_overview', 'OPTION - SWAF before Overview', False)
546 GuidedAcqDialog.AddIntegerRange('SWAF_ov_initial_range', 'Initial SWAF Range before Overview [micron]', 200, 50, 3000)
547 GuidedAcqDialog.AddLabel('------   Select Image Analysis to detect objects   ------')
548 GuidedAcqDialog.AddDropDown('ip_pipe', 'Image Analysis Pipeline', ipfiles_short, 0)
549 GuidedAcqDialog.AddCheckbox('use_poly', 'Use Polygons from ZEN IA instead of BBox', True)
550 GuidedAcqDialog.AddCheckbox('use_apeer_for_IA', 'Use APEER module run run IA instead', False)
551 GuidedAcqDialog.AddDropDown('ap_pipe', 'APEER Module Settings', apfiles_short, 0)
552 GuidedAcqDialog.AddLabel('------   Select DetailScan Experiment   ------')
553 GuidedAcqDialog.AddDropDown('detailed_exp', 'Detailed Scan Experiment', expfiles_short, 1)
554 GuidedAcqDialog.AddCheckbox('fs_before_detail', 'OPTION - FindSurface (DF only) before Detail', False)
555 GuidedAcqDialog.AddCheckbox('SWAF_before_detail', 'OPTION - SWAF before Detail', False)
556 GuidedAcqDialog.AddIntegerRange('SWAF_detail_initial_range', 'Initial SWAF Range before Detail [micron]', 100, 10, 1000)
557 GuidedAcqDialog.AddCheckbox('recallfocus_beforeDT', 'OPTION - Use RecallFocus (DF only) before Detail', False)
558 GuidedAcqDialog.AddLabel('------   Specify basefolder to save the images   ------')
559 GuidedAcqDialog.AddFolderBrowser('outfolder', 'Basefolder for Images and Data Tables', imgfolder)
560
561 # show the window
562 result = GuidedAcqDialog.Show()
563 if result.HasCanceled:
564     message = 'Macro was canceled by user.'
565     print(message)
566     raise SystemExit
567
568 # get the values and store them
569 OverViewExpName = str(result.GetValue('overview_exp'))
570 ImageAS = str(result.GetValue('ip_pipe'))
571 ApeerMS = str(result.GetValue('ap_pipe'))
572 DetailExpName = str(result.GetValue('detailed_exp'))
573 OutputFolder = str(result.GetValue('outfolder'))
574 fs_beforeOV = result.GetValue('fs_before_overview')
575 SWAF_beforeOV = result.GetValue('SWAF_before_overview')
576 SWAF_beforeOV_range = result.GetValue('SWAF_ov_initial_range')
577 fs_beforeDT = result.GetValue('fs_before_detail')
578 SWAF_beforeDT = result.GetValue('SWAF_before_detail')
579 SWAF_beforeDT_range = result.GetValue('SWAF_detail_initial_range')
580 RecallFocus = result.GetValue('recallfocus_beforeDT')
581 use_apeer = result.GetValue('use_apeer_for_IA')
582 use_polygon = result.GetValue('use_poly')
```

The resulting dialog window is shown below.

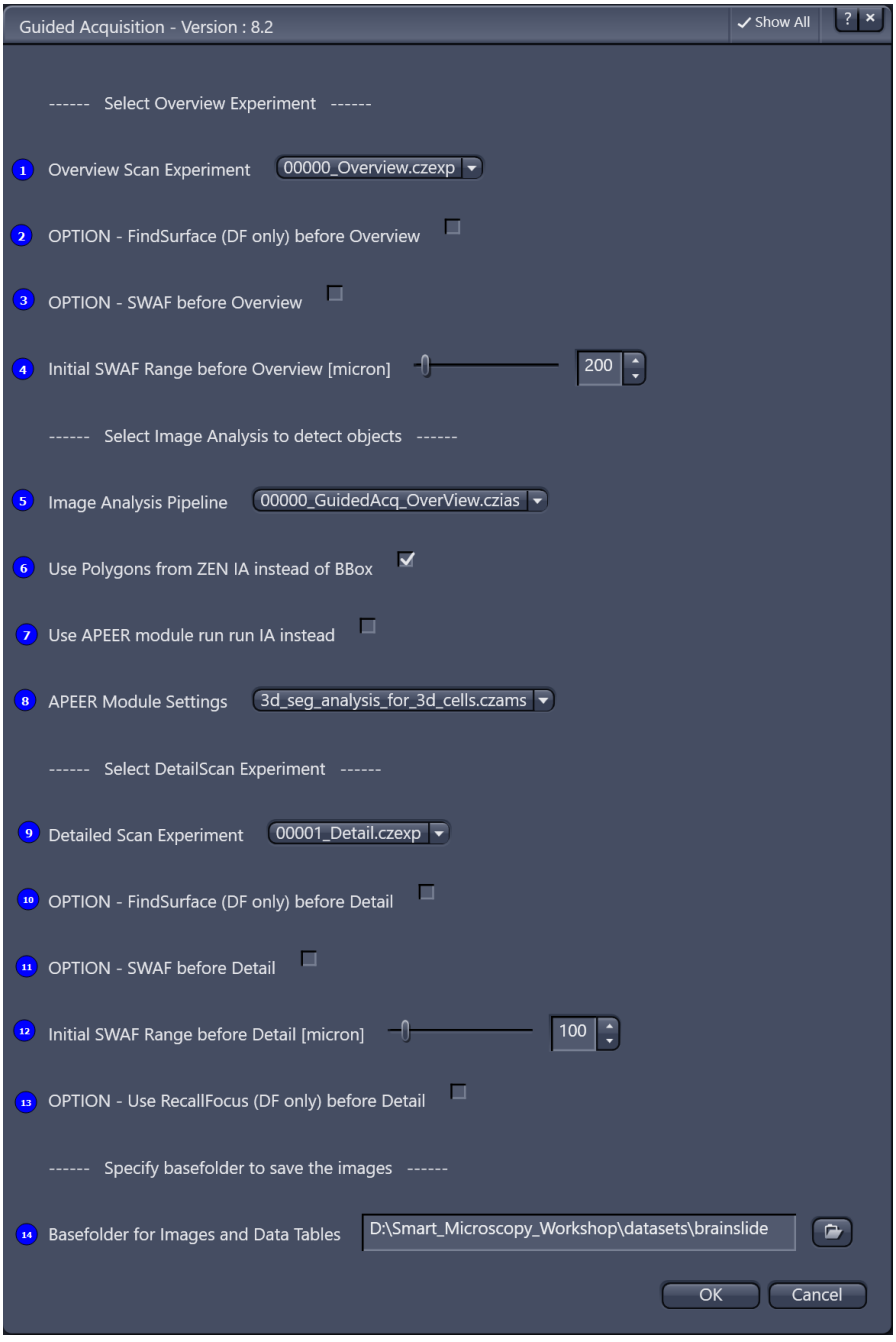

Figure 9: Simple User Dialog to run the Guided Acquisition WorkFlow (Remark: Version in screenshot may differ from the actual latest version)

3.5 Overview Scan Experiment

At this point we have all the information we need to actually start the workflow. The first step is to execute the actual overview scan experiment, which is most likely a tile experiment. The resulting image will be saved to the specified folder.

```
############ START OVERVIEW SCAN EXPERIMENT #################

if fs_beforeOV:
    # initial focussing via FindSurface to assure a good starting position
    Zen.Acquisition.FindSurface()
    print('Z-Position after FindSurface: ', Zen.Devices.Focus.ActualPosition)

if SWAF_beforeOV:
    zSWAF = runSWAF_special(OVScan_reloaded,
                            delay=hwdelay,
                            searchStrategy='Full',
                            sampling=ZenSoftwareAutofocusSampling.Coarse,
                            relativeRangeIsAutomatic=False,
                            relativeRangeSize=SWAF_beforeOV_range,
                            timeout=1)

# get the resulting z-position
znew = Zen.Devices.Focus.ActualPosition

# adapt the Overview Scan Tile Experiment with new Z-Position
if OVScanIsTileExp:
    OVScan_reloaded.ModifyTileRegionsZ(blockindex, znew)
    print('Adapted Z-Position of Tile OverView. New Z = ', '%.2f' % znew)

# execute the experiment
print('\nRunning OverviewScan Experiment.\n')
#output_OVScan = Zen.Acquisition.Execute(OVScan_reloaded)
# For testing purposes - Load overview scan image automatically instead of executing the "real" experiment
ovtestimage = r'C:\Users\m1srh\OneDrive - Carl Zeiss AG\Smart_Microscopy_Workshop\datasets\brainslide\OverViewScan.czi'
output_OVScan = Zen.Application.LoadImage(ovtestimage, False)

# the the stage top-left and the scaling of the overview image
stageTL = output_OVScan.GetPositionLeftTop()
ovscaling = output_OVScan.Scaling

# show the overview scan inside the document area
Zen.Application.Documents.Add(output_OVScan)
ovdoc = Zen.Application.Documents.GetByName(output_OVScan.Name)

# save the overview scan image inside the select folder
savepath_ovscan = Path.Combine(OutputFolder, ovscan_name)
output_OVScan.Save(savepath_ovscan)

# get the actual focus value for the overscan independent from the z-value
# before the start of the overview scan using the image metadata
zvalue_ovscan = output_OVScan.Metadata.FocusPositionMicron

############ END OVERVIEW SCAN EXPERIMENT ##################
```

A very important part of such an automated workflow is to find the focus. To ensure this, the desired objective and optovar are changed before the overview starts, followed by a series of optional focus actions:

1. Find the surface of the sample carrier.
2. Focus onto the specimen sample via SWAF.
3. Store the resulting new Z-position.

When testing out applications like this it is very useful to use test data sets to save time instead of re-running an acquisition many times. Therefore it can be helpful to comment the lines where the real experiment is executed and uncomment the line inside the script, where one can load an already acquired overview image.

3.6 Find the interesting Objects

Now it is time to run the image analysis on the overview image. As a result, two tables will be created inside ZEN. The first contains information about all objects and the second contains information about the single objects (for instance their stage positions). This second table is what will be used as an input for the detailed scan later on.

```
# run normal ZEN image analysis
if not use_apeer:

    # Load analysis setting created by the wizard or an separate macro
    ias = ZenImageAnalysisSetting()

    # for simulation use: 000 - RareEventExample.czias
    ias.Load(ImageAS)

    # Analyse the image
    Zen.Analyzing.Analyze(output_OVScan, ias)

    # get classes and derive regions names from that
    iasclasses = getclassnames(ias)

    if use_polygon:

        # get the individual image analysis regions from 1st IA class (!!!)

        # ATTENTION: the first single objects class will be used. It has ID = 2
        regions = Zen.Analyzing.GetRegions(output_OVScan, iasclasses['2'])
        print('RegionClassNames: ', iasclasses)
        print('Analysis found ' + str(regions.Count) + 'regions!')

        # create dictionary for polygon regions
        polyregions = {}

        # loop over all regions and get the points of polygon (outline of the object)
        for i in range(0, regions.Count):
            points = regions[i].GetPolygon()

            # convert points to stage coordinates
            polyregions[i] = create_exppolygon(points, stageTL, ovscaling)

    # Create Zen table with results for all detected objects (parent class)
    AllObj = Zen.Analyzing.CreateRegionsTable(output_OVScan)

    # Create Zen table with results for each single object
    SingleObj = Zen.Analyzing.CreateRegionTable(output_OVScan)

    # check for existence of required column names inside table
    soi = SingleObj.GetBoundsColumnInfoFromImageAnalysis(True)

    # 1st item is a bool indicating if all required columns could be found
    columnsOK = soi.AreRequiredColumnsAvailabe

    if not columnsOK:
        print('Execution stopped. Required Columns are missing.')
        raise Exception('Execution stopped. Required Columns are missing.')

    # show and save data tables to the specified folder
    Zen.Application.Documents.Add(AllObj)
    Zen.Application.Documents.Add(SingleObj)
    AllObj.Save(Path.Combine(OutputFolder, 'OverviewTable.csv'))
    SingleObj.Save(Path.Combine(OutputFolder, 'SingleObjectsTable.csv'))

# use APEER module to detect the objects
if use_apeer:

    # read it from settings file
    ams = ZenApeer.Onsite.ModuleSetting()

    # loaf the APEER module seeting b - remove *.czams file extension first
    ams.Load(Path.GetFileNameWithoutExtension(ApeerMS))

    print('-----   Apeer Module Setting   -----')
    print('Module Name    : ', ams.ModuleName)
    print('Module Version : ', ams.ModuleVersion)
    print('Module Parameters    : ', ams.Parameters)

    # get module and check
    mymodule, version_found = get_module(ams.ModuleName, module_version=ams.ModuleVersion)

    # exit if the check failed
```

```
    if mymodule is None or not version_found:
        print('Module check failed. Exiting.')
        raise SystemExit

    # get the module parameters for the specified module
    module_params = ZenApeer.Onsite.GetSampleModuleParameters(ams.ModuleName, ams.ModuleVersion)

    # get the module input programmatically
    module_inputs = get_module_inputs(module_params)

    # create the required dictionary with the correct key and value
    input_image = {module_inputs[0]: savepath_ovscan}

    # create the path to save the results
    #savepath_apeer = Path.Combine(OutputFolder, 'apeer_results')
    savepath_apeer = OutputFolder

    # create the output directory if not existing already
    if not Directory.Exists(savepath_apeer):
        Directory.CreateDirectory(savepath_apeer)

    # run the local APEER module with using keywords
    try:
        runoutputs, status, log = ZenApeer.Onsite.RunModule(moduleName=ams.ModuleName,
                                                            moduleVersion=ams.ModuleVersion,
                                                            inputs=input_image,
                                                            parameters=ams.Parameters,
                                                            storagePath=savepath_apeer)

        for op in runoutputs.GetEnumerator():
            print('-----    Outputs   -----')
            print(op.Key, ' : ', op.Value)

    except ApplicationException as e:
        print('Module Run failded.', e.Message)
        raise SystemExit

    # get results storage locations
    # IMPORTANT: To be used inside Guided Acquisition
    # the APEER Module is expected to have
    # at least those to outputs with exactly those names

    if not runoutputs.ContainsKey('segmented_image'):
        print('No output : segmented_image')
        raise SystemExit

    if not runoutputs.ContainsKey('objects_table'):
        print('No output : objects_table')
        raise SystemExit

    # load the segmented image and make sure the pyramid is calculated
    segmented_image = Zen.Application.LoadImage(runoutputs['segmented_image'], False)
    Zen.Processing.Utilities.GenerateImagePyramid(segmented_image, ZenBackgroundMode.Black)
    Zen.Application.Documents.Add(segmented_image)

    # auto-display min-max
    ids = segmented_image.DisplaySetting.GetAllChannelIds()
    for id in ids:
        segmented_image.DisplaySetting.SetParameter(id, 'IsAutoApplyEnabled', True)

    # initialize ZenTable object and load CSV file
    SingleObj = ZenTable()
    SingleObj.Load(runoutputs['objects_table'])
    Zen.Application.Documents.Add(SingleObj)

    # get all columns as dict with columnIDs
    colID = get_columns(SingleObj)

    # define the required column names here
    col2check = ('bbox_center_stageX', 'bbox_center_stageY', 'bbox_width_scaled', 'bbox_height_scaled')

    # check if all columns exist
    if all(key in colID for key in col2check):
        print('All required columns found.')
    else:
        print('Not All required columns found. Exiting.')
        raise SystemExit

# check the number of detected objects = rows inside image analysis table
num_POI = SingleObj.RowCount
```

As shown in figure 6, it is required to have the correct image analysis features selected. Only the will the respective columns inside the table exist. This has to be checked because otherwise the workflow will be impossible.

The "checking" itself is again done by a tool function. The output will be a boolean and a dictionary in order to look up the column index for the respective image analysis parameters. This has the advantage of being independent from a specific column order. One just has to make sure that all required columns are there.

If the option inside the initial dialog window was checked, the offset between the sample carrier surface and the actual specimen will be measured and stored inside the Definite Focus via Store Focus, so that Recall Focus can be used later on when needed. The dialog also offers the possibility to enter an arbitrary offset manually, which will only be applied if the offset was not determined automatically.

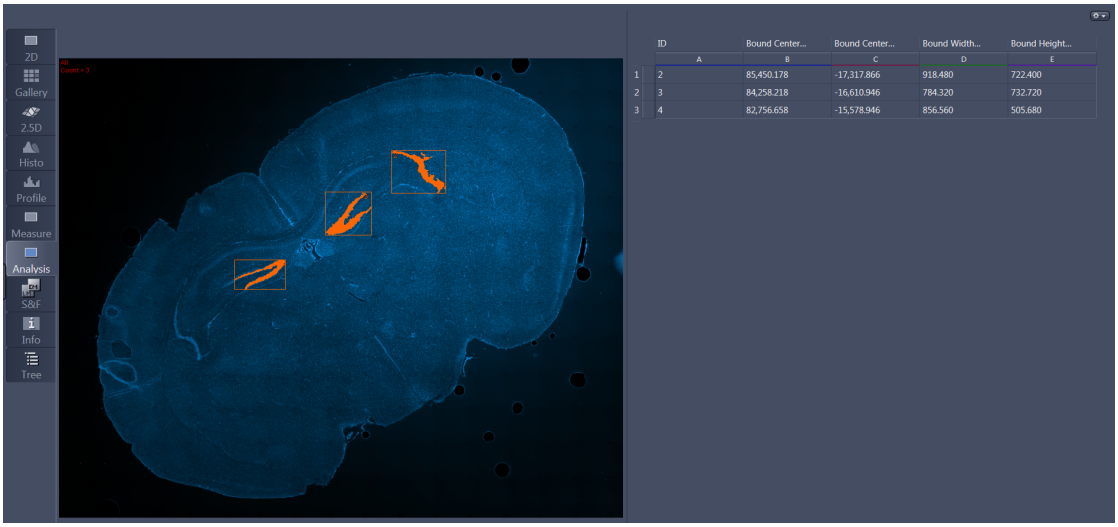

Figure 10: The results of the overview scan and the image analysis.

Finally the number of detected objects is checked by looking up the number of rows inside the table containing the information about the single objects.

3.7 Prepare Detail Scan

Before the actual loop over all objects can start one has to do the following steps:

1. Move stage to the 1st XY position from the object table
2. Move z-Drive to the actual focus position of the overview scan

3.8 Modify the Tile Dimensions

Once one has the list of objects with the stage coordinates, the next task is to create a loop to cycle through all detected positions.

- Move to the correct XYZ stage positions.
- Run Focus Actions when required.
- Initialize the detail scan experiment at the new position.
- Adapt TileRegion using a BoundingBox or ...
- ... modify TileRegion using the Polygon Points.

```
############# START DETAILED SCAN EXPERIMENT #############

# again get the resulting focus position
zpos = Zen.Devices.Focus.ActualPosition

if not use_apeer:

    # check for the column 'ID' which is required
    ID_exists, column_ID = checktableentry(SingleObj, entry2check='ID')

    # execute detailed experiment at the position of every detected object
    for obj in range(SingleObj.RowCount):

        # get the object information from the position table
        POI_ID = SingleObj.GetValue(obj, column_ID)

        # get XY-stage position from table
        xpos = SingleObj.GetValue(obj, soi.CenterXColumnIndex)
        ypos = SingleObj.GetValue(obj, soi.CenterYColumnIndex)

        # move to the current position
        Zen.Devices.Stage.MoveTo(xpos + dx_detector, ypos + dy_detector)
        print('Moving Stage to Object ID:', POI_ID, ' at :', '%.2f' % xpos, '%.2f' % ypos)

        # try to apply RecallFocus (DF only) when this option is used
        if userecallfocus:
            zpos = apply_recall_focus()

        # if DetailScan is a Tile Experiment
        if DetailIsTileExp:

            print('Detailed Experiment contains TileRegions.')
            # Modify tile center position - get bounding rectangle width and height in microns
            bcwidth = SingleObj.GetValue(obj, soi.WidthColumnIndex)
            bcheight = SingleObj.GetValue(obj, soi.HeightColumnIndex)

            if use_polygon:
                # modify the experiment with th respective polygon region
                DetailScan_reloaded = modify_tileexperiment_polygon(DetailScan_reloaded, polyregions[obj],
                                                                    blockindex=0,
                                                                    zpos=zpos)
            else:
                # modify the experiment
                DetailScan_reloaded = modify_tileexperiment_rect(DetailScan_reloaded,
                                                                 blockindex=blockindex,
                                                                 bcwidth=bcwidth,
                                                                 bcheight=bcheight,
                                                                 xpos=xpos,
                                                                 ypos=ypos,
                                                                 zpos=zpos)

        if not DetailIsTileExp:
            print('Detailed Experiment does not contains TileRegions. Nothing to modify.')
```

3.9 Run the Detailed Scan

The last steps, which need to be executed are:

- Run the (modified) detailed scan experiment here.
- Save the data to the specified folder.
- Close the image document in ZEN.
- Rename the file using the current object ID.

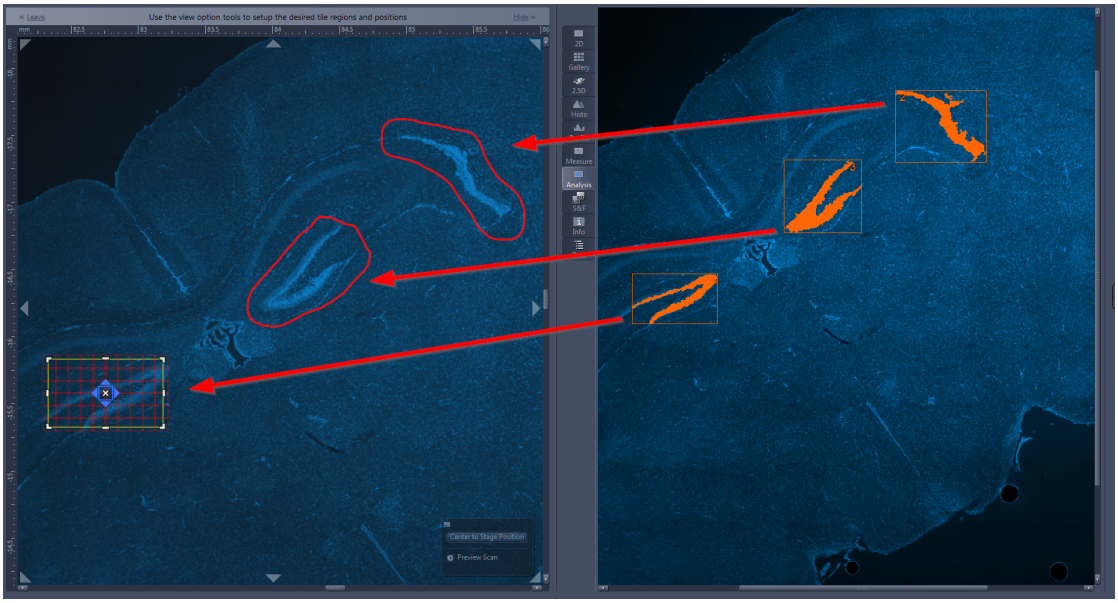

Figure 11: The results of the detailed scan - tile regions adapts to objects.

That is it. All image data sets acquired at the detected positions are stored inside the correct folder using the object IDs as names. The tables are saved in the same location.

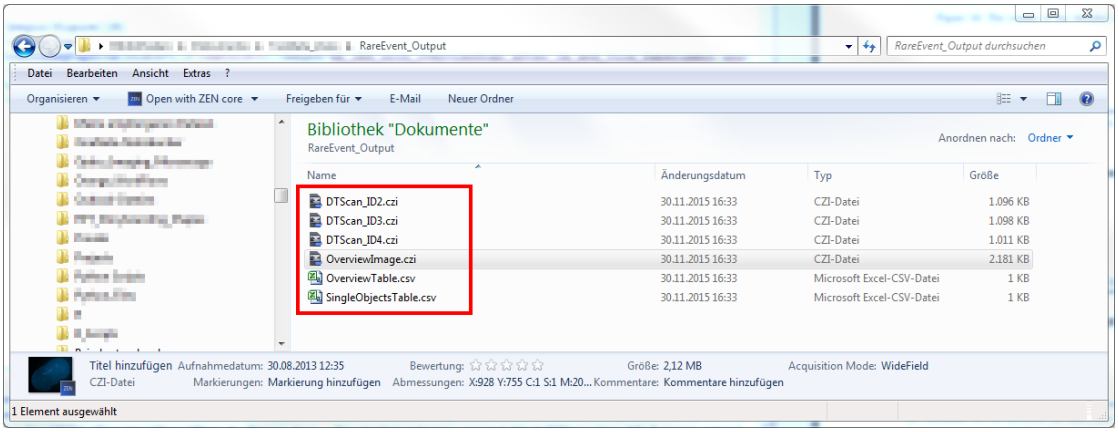

Figure 12: The results of the detailed scan stored inside a folder.

4 ZEN Blue Software Module for Guided Acquisition

Starting with ZEN Blue 3.1 there will the option to purchase dedicated software module for Guided Acquisition, which has a streamlined UI interface and other additional functionality, like save settings and pre-processing functions etc.

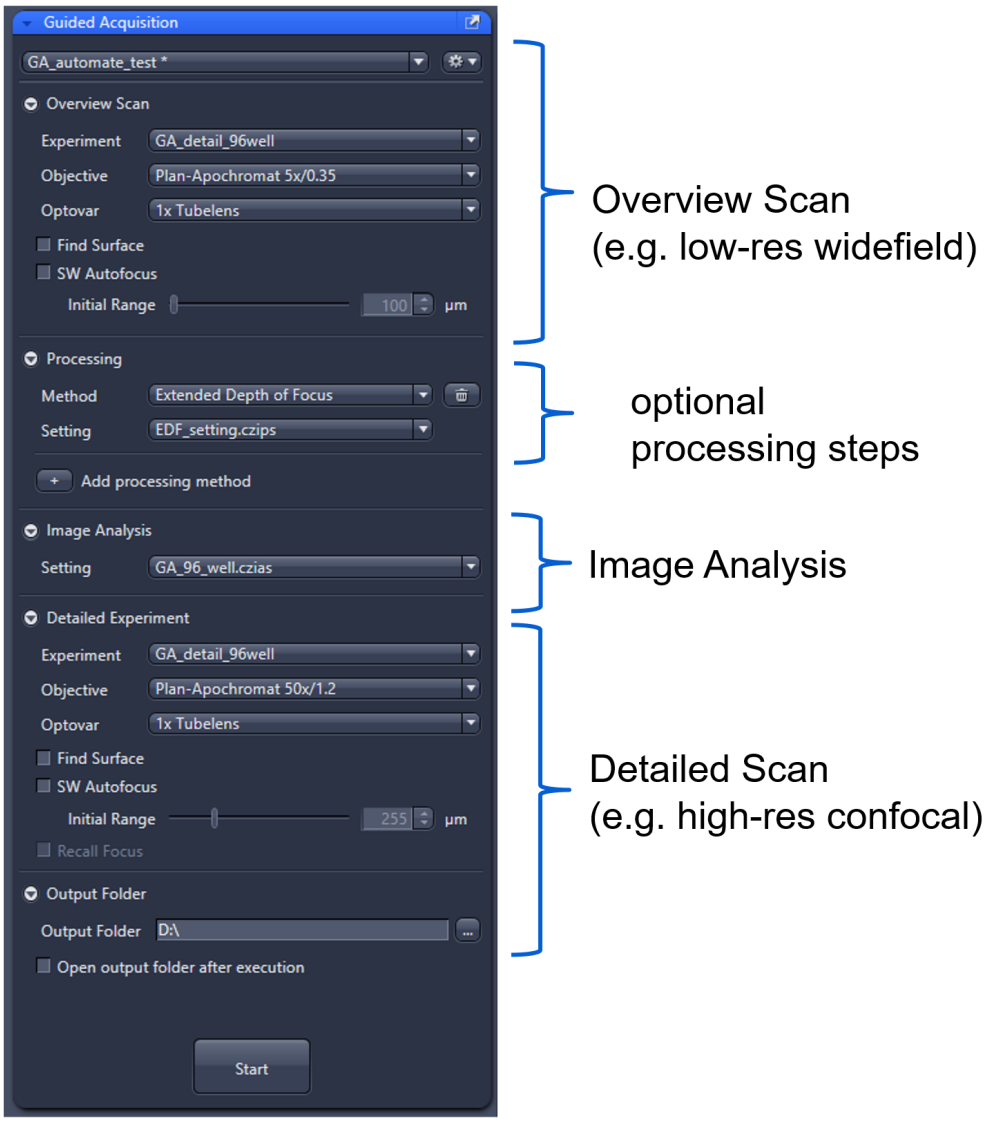

Figure 13: Software Module - Guided Acquisition (ZEN Blue 3.1 or better)

5 Testrun with FluoCells Slide

To test the workflow on a real sample one can use the FluoCells slide. The general idea is to setup an image analysis that detects some "interesting" objects. In order to create a somewhat short test run it is recommended to use an image analysis pipeline that only yields in a few positive detection events.

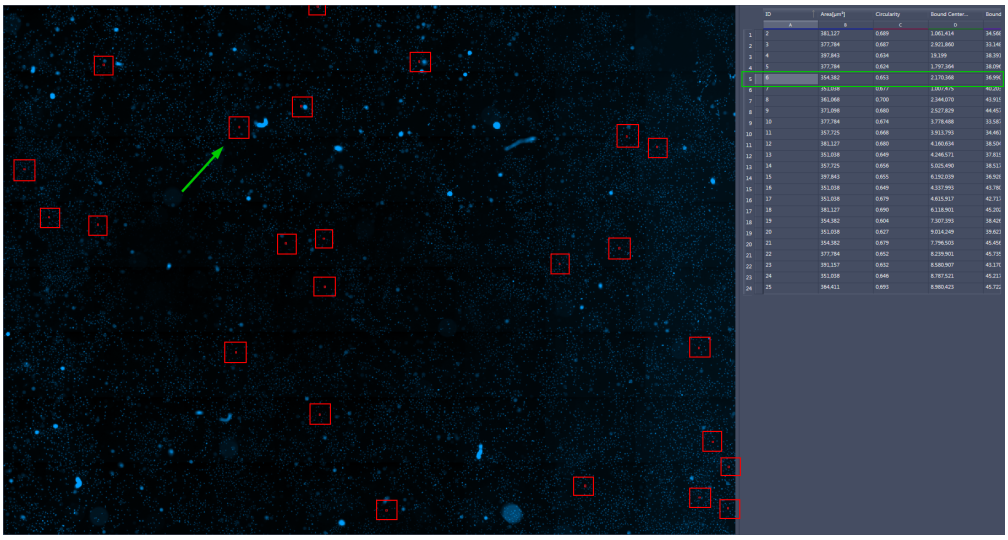

Figure 14: The results of the overview scan and image analysis.

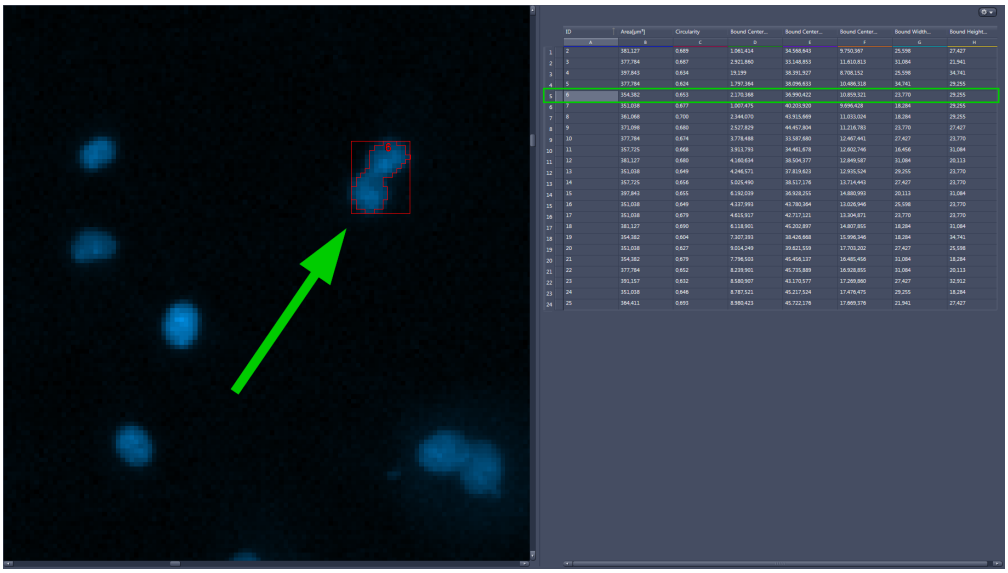

Figure 15: Detailed view on a detected object.

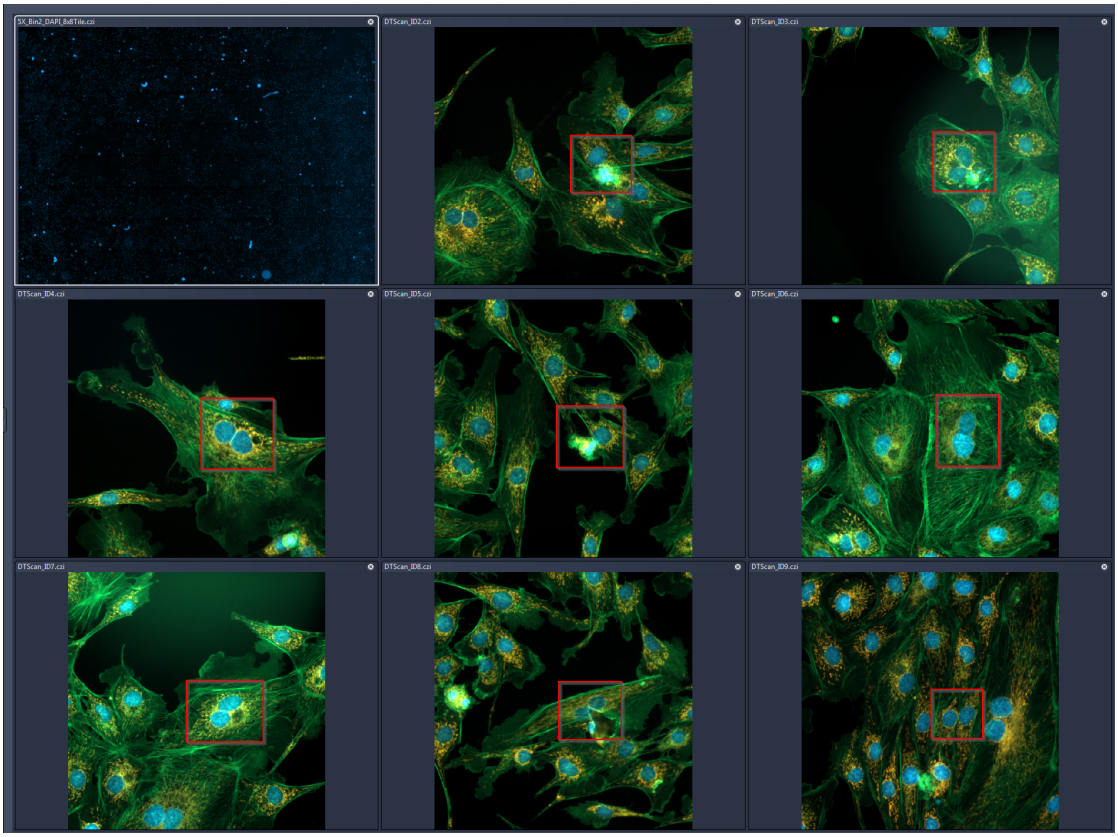

Figure 16: Detailed images from some of the detected objects.

The image analysis yielded 24 positive detection of cells with a large nucleus, preferably cells which were "caught" during cell division. Keep in mind, that the image analysis was not "fine-tuned" to catch all cell divisions. It is all about testing out the general workflow. For this reason it is also good practice to acquire a small overview scan at the beginning, e.g. a 2x2 tile image.

6 Test Run with Brain Slide

The same basic workflow can be also used to detect brain slices on a slide. All one needs to change is the image analysis pipeline in order to detect the slices.

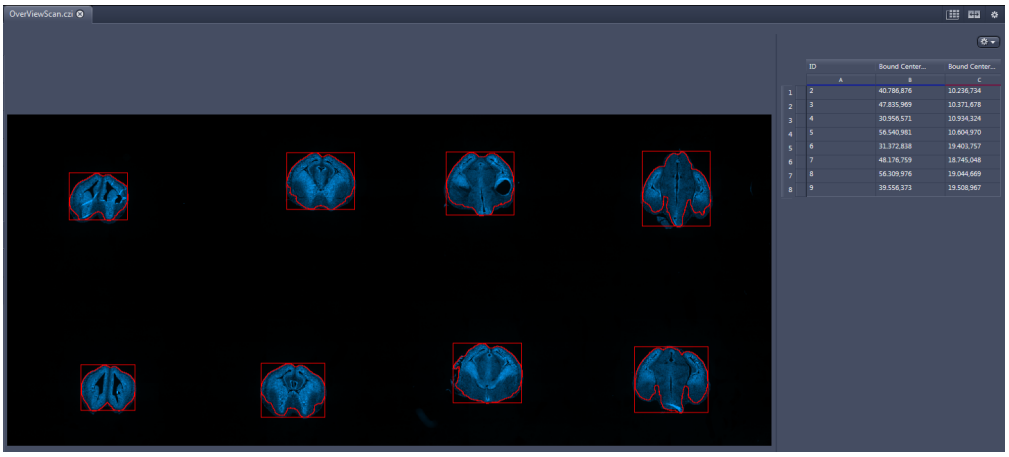

Figure 17: The results of the overview scan and image analysis.

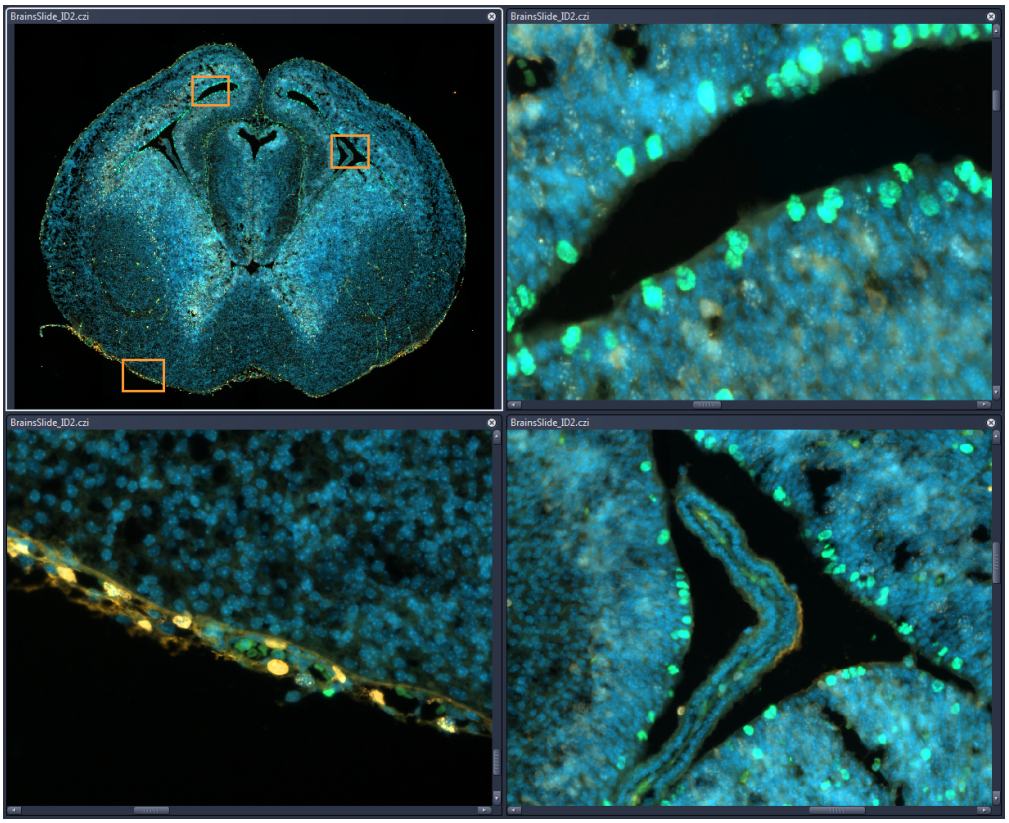

Figure 18: A more detailed image of the first specimen with three zoomed-in regions.

The image analysis yielded eight positive regions for brains slices on this slide. The properties of the bounding rectangle were used to modify the required tile experiment. The final detailed scan contained 36 tiles and it was imaged using only a hardware-based focus system.

Keep in mind that the actual focusing during such a experiment must be still part of the focus strategies with the Detail Scan experiment. Which one must be used here greatly depends on the nature of the sample and the actual application. The Guided Acquisition tool only supports finding the initial positions based on the overview scan image.

7 Mitosis Detection with Camera and LSM

An especially interesting option is to combine the power of a camera-based overview scan with the optional sectioning capabilities of an LSM. Such an workflow can be easily configured in ZEN Blue by setting up the respective experiment for the overview scan with camera detection using a low magnification and the LSM-based detail scan, e.g. Z-Stack, using a high NA objective.

Since ZEN Blue 2.5 this software has a module called ZEN Connect, which allows combining and correlating images inside one sample-centric workspace. Every acquired image by either the camera or the LSM) will be placed here based on the XY stage coordinates. Therefore the Correlative Workspace (CWS) is ideally suited to display the results of an Guided Acquisition workflow.

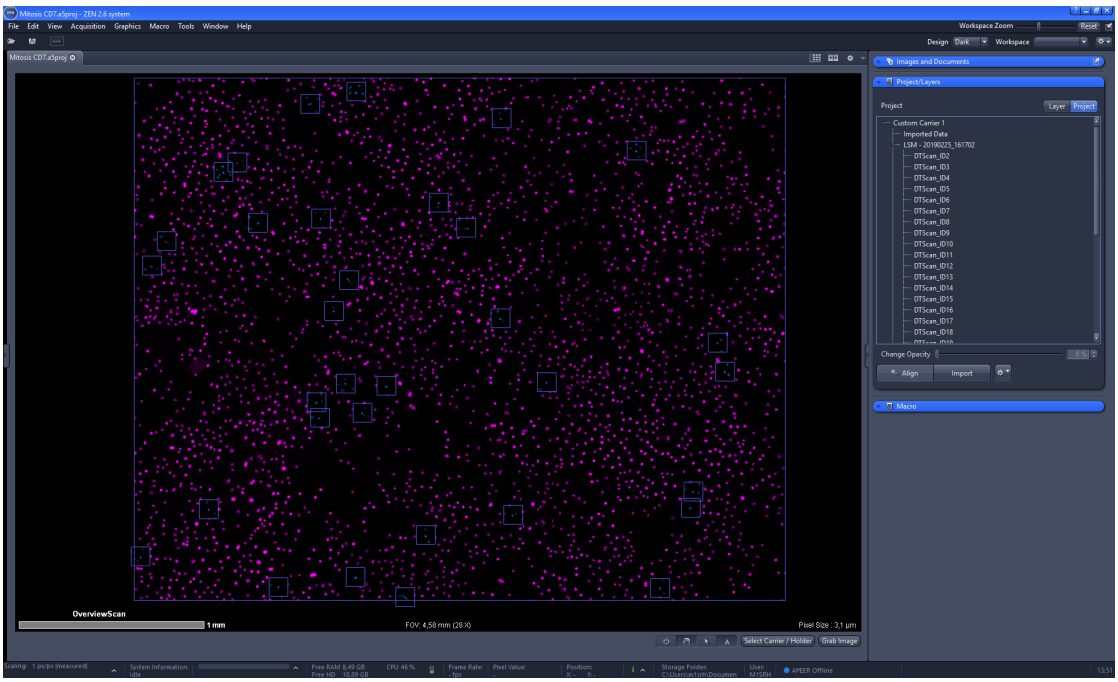

Figure 19: The results of the overview scan and image analysis inside the CWS.

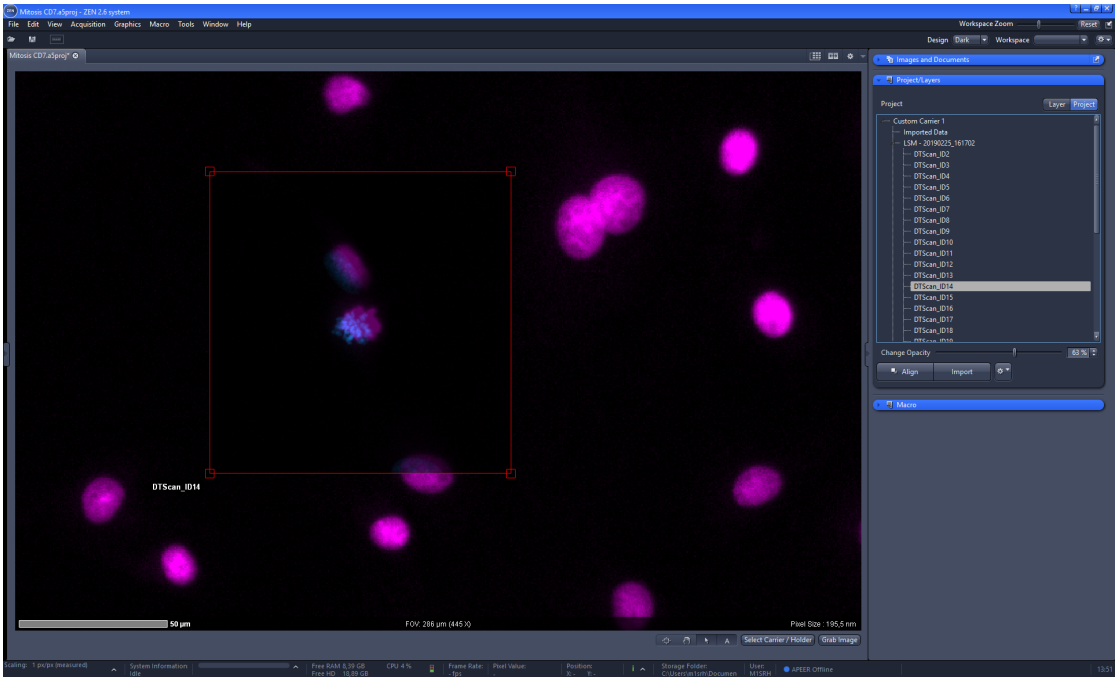

Figure 20: Overlay between positive detection inside overview scan and detailed scan using the Correlative Workspace (CWS).

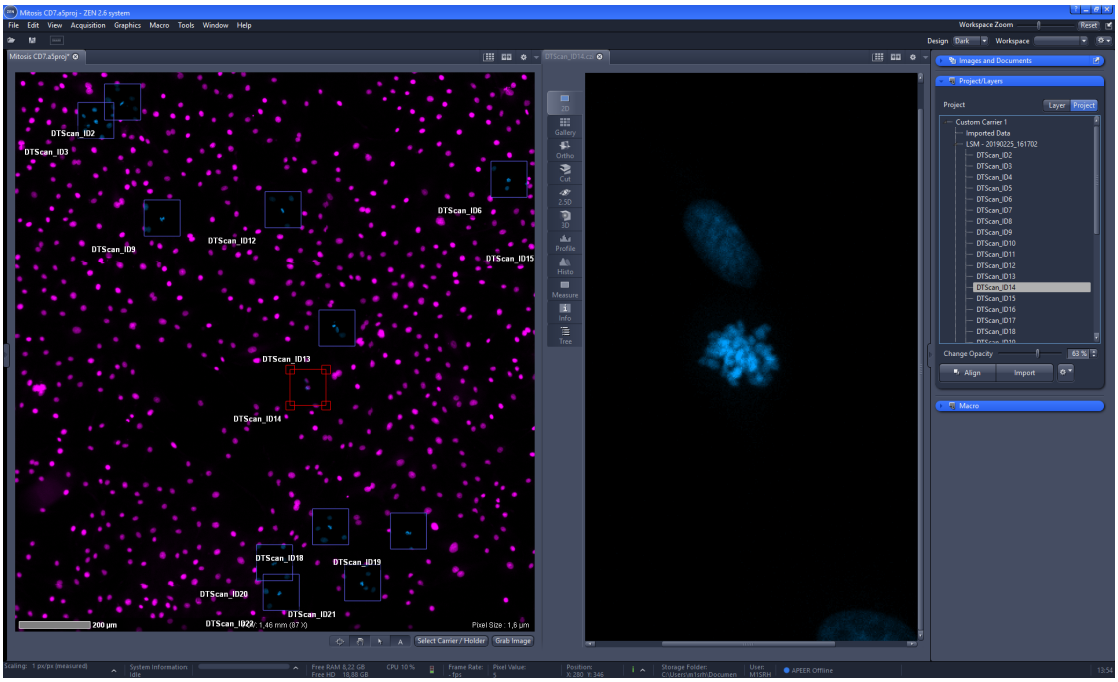

Figure 21: Side-by-Side view of the CWS and the normal view inside ZEN Blue.

8 Appendix

8.1 Complete Workflow Diagram

This is the complete workflow diagram for Guided Acquisition.

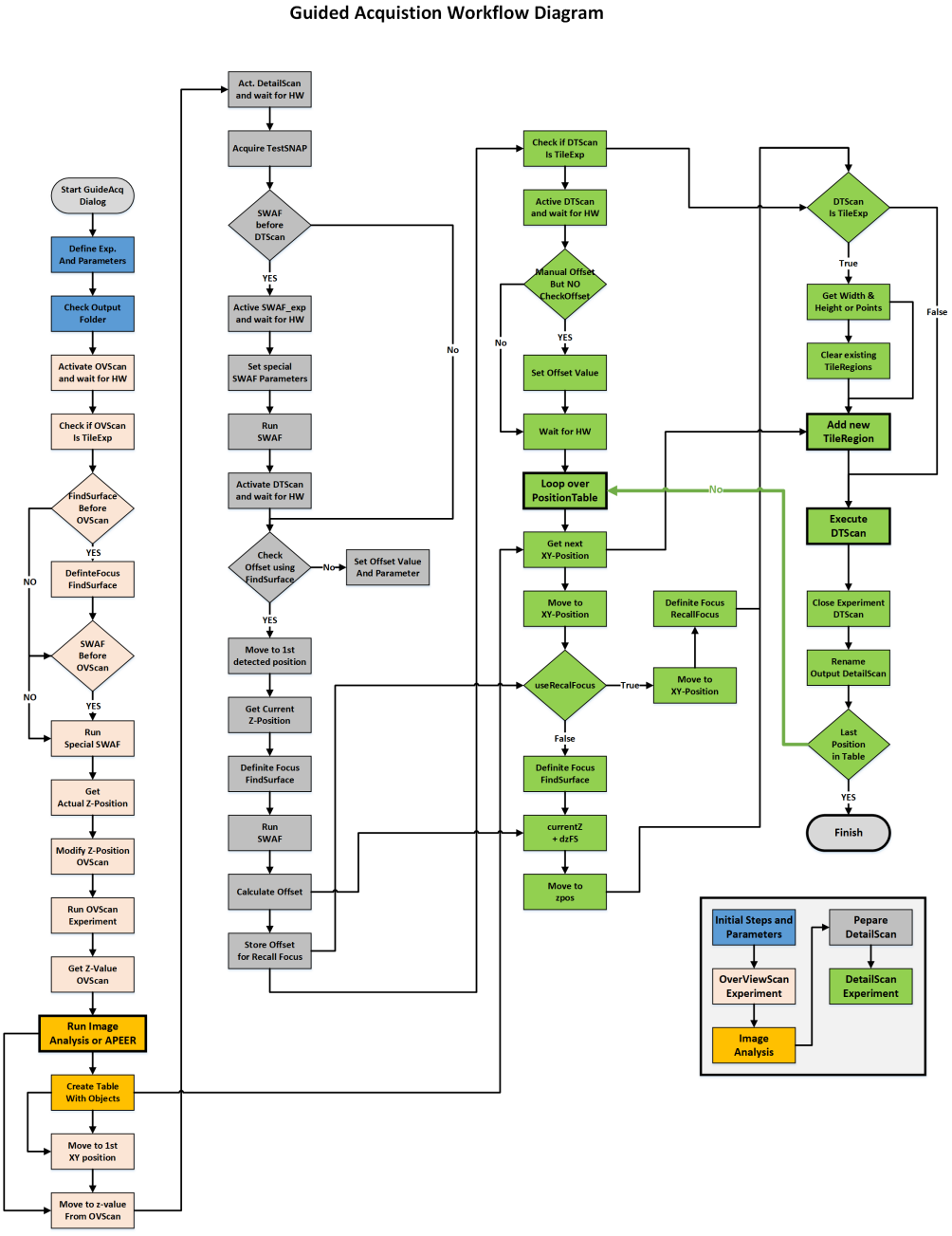

Figure 22: Complete workflow diagram for Guided Acquisition

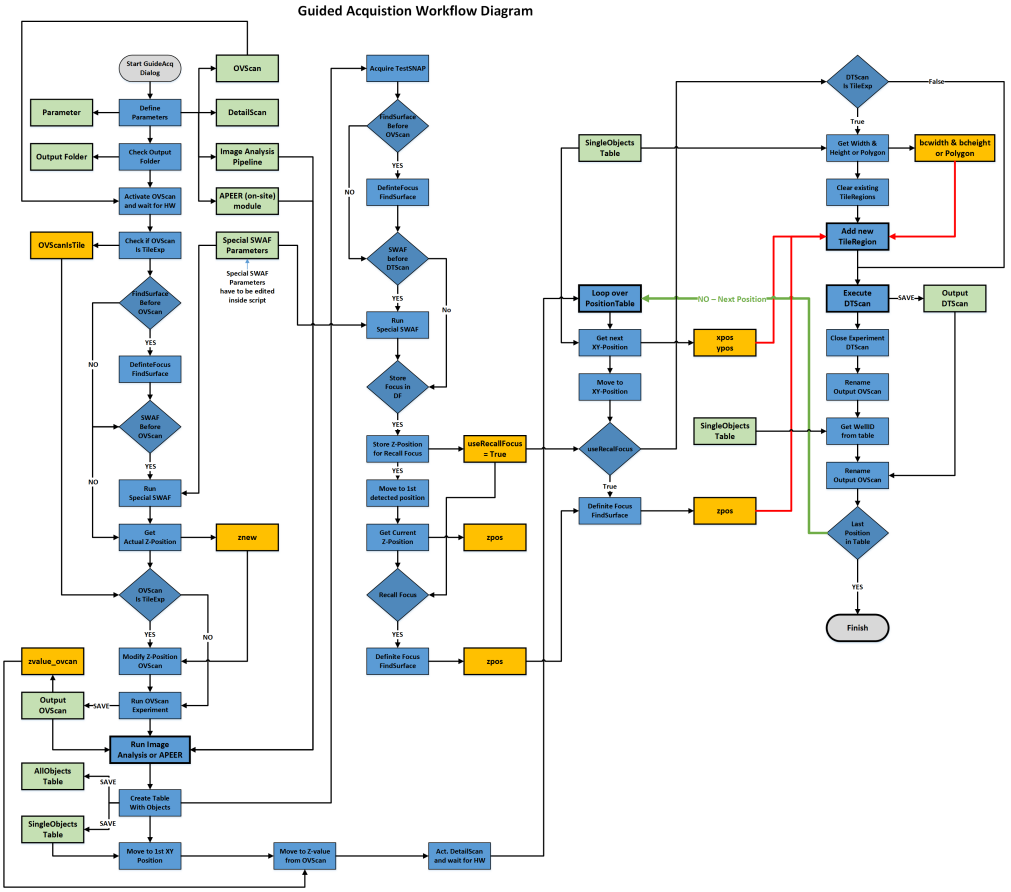

Figure 23: Complete workflow diagram for Guided Acquisition

8.2 Python Script: Guided_Acquistion_shortUI.py

This the the complete ZEN Blue python script. It can work on any motorized microscope stand using ZEN blue (with some modifications), but it was tested and optimized mainly for the Celldiscoverer 7 and for the AxioObserver 7. Both of those stands have the Definite Focus 2 as an additional hardware option, which is strongly advised for such highly automated workflows, where fast and efficient focus options, such as Find Surface are really a big plus.

```
1  ################################################################
2  # File        : Guided_Acquisition_shortUI.py
3  # Version     : 8.2
4  # Author      : czsrh, czmla, czkel
5  # Date        : 14.06.2021
6  # Institution : Carl Zeiss Microscopy GmbH
7  #
8  # Optimized for the use with Celldiscoverer 7 and DF2, but
9  # applicable for all motorized stands running in ZEN Blue.
10 # Please adapt focussing commands, especially FindSurface
11 # when using with other stands.
12 #
13 # 1) - Select Overview Scan Experiment
14 # 2) - Select appropriate Image Analysis Pipeline or APEER module setting
15 #      to detect objects of interest
16 # 3) - Select Detailed Scan Experiment
17 # 4) - Specify the output folder for the image and data tables
18 #
19 # Tested with ZEN blue 3.4.
20 # Should work also with later version with some limitations
21 #
22 # Disclaimer: This tool is purely experimental. Feel free to
23 # use it at your own risk. Especially be aware of the fact
24 # that automated stage movements might damage hardware if
25 # one starts an experiment and the the system is not setup
26 # and calibrated properly. Check everything in simulation mode first!
27 #
28 # Copyright(c) 2021 Carl Zeiss AG, Germany. All Rights Reserved.
29 #
30 # Permission is granted to use, modify and distribute this code,
31 # as long as this copyright notice remains part of the code.
32 ################################################################
33
34 import time
35 from datetime import datetime
36 import errno
37 from System import Array
38 from System import ApplicationException
39 from System import TimeoutException
40 from System.IO import File, Directory, Path
41 from System.Collections.Generic import List
42 import sys
43
44 # version number for dialog window
45 version = 8.2
46 # file name for overview scan
47 ovscan_name = 'OverviewScan.czi'
48
49 """
50 Additional XY offest for possible 2nd port relative to the 1st port
51 Example: 1st port Camera and 2nd port LSM
52 Can be set to Zero, when system is correctly calibrated
53 !!! Only use when overview and detailed scan are using different detector !!!
54 """
55 dx_detector = 0.0
56 dy_detector = 0.0
57
58 # experiment blockindex
59 blockindex = 0
60 # delay for specific hardware movements in [seconds]
61 hwdelay = 1
62 # posprocessing switch
63 do_postprocess = False
64
65 ################################################################
66
67
68 def find_module(name):
69     """Finds an APEER module given its Name.
70
71     :param name: The name of the module to look for
72     :type name: str
```

```
    :return: The first module with the specified name or None, if no such module exists.
    :rtype: ApeerModule
    """

    modules = ZenApeer.Onsite.ListLocalModules()

    return next((m for m in modules if m.ModuleName == name), None)


def get_module_inputs(params):
    """Get the module inputs.

    :param params: Apeer module parameters
    :type params: ApeerModuleDescription
    :return: List with input parameters
    :rtype: list
    """
    module_inputs = []
    for ip in params.Inputs:
        module_inputs.append(ip.Key)

    return module_inputs


def get_columns(table):
    """Get columns names and their IDs as a dictionary.

    :param table: Input table
    :type table: ZenTable
    :return: Dictionary with columns names and their respective IDs
    :rtype: dict
    """
    col_dict = {}
    colid = -1
    # loop over all columns
    for col in range(0, table.ColumnCount):
        colid += 1
        # get the caption and store in dictionary with ID as value
        colcaption = table.Columns[col].Caption
        col_dict[colcaption] = colid

    return col_dict


def get_module(module_name, module_version=0):
    """Get an APEER module and check if the desired version is available.

    :param module_name: APEER module name
    :type module_name: str
    :param module_version: Version number of the APEER module, defaults to 0
    :type module_version: int, optional
    :return: mymodule, version_found
    :rtype: ApeerModule, bool
    """
    version_found = False

    # try to find the desired module
    mymodule = find_module(module_name)

    if mymodule is not None:
        print 'Module : ' + module_name + ' found.'
    elif mymodule is None:
        print 'Module : ' + module_name + ' not found.'

    if str(module_version) in mymodule.AvailableVersions:
        print 'Module : ' + module_name + ' Version ' + str(module_version) + ' found.'
        version_found = True
    elif not str(module_version) in mymodule.AvailableVersions:
        print 'Module : ' + module_name + ' Version ' + str(module_version) + ' not found.'

    return mymodule, version_found


def getshortfiles(filelist):
    """Create list with shortended filenames

    :param filelist: List with files with complete path names
    :type filelist: list
    :return: List with filenames only
    :rtype: list
    """
    files_short = []
    for short in filelist:
        files_short.append(Path.GetFileName(short))

    return files_short


def checktableentry(datatable, entry2check='ImageSceneContainerName'):
```

```
    """Check ZEN table for an existing column by name.

    :param datatable: ZEE Table with data.
    :type datatable: ZenTable
    :param entry2check: Column to look for, defaults to 'ImageSceneContainerName'
    :type entry2check: str, optional
    :return: entry_exits, column
    :rtype: bool, integer
    """

    num_col = datatable.ColumnCount
    entry_exists = False
    column = None

    for c in range(0, num_col):
        # get the current column name
        colname = datatable.Columns[c].ColumnName
        if colname == entry2check:
            column = c
            entry_exists = True
            break

    return entry_exists, column


def dircheck(basefolder):
    """Check if a directory or folder already exits.
    Create the folder if it does not exist

    :param basefolder: folder to check
    :type basefolder: str
    :return: new_directory
    :rtype: str
    """

    # check if the destination basefolder exists
    base_exists = Directory.Exists(basefolder)

    if base_exists:
        print('Selected Directory Exists: ', base_exists)
        # specify the desired output format for the folder, e.g. 2017-08-08_17-47-41
        format = '%Y-%m-%d_%H-%M-%S'

        # create the new directory
        newdir = createfolder(basefolder, formatstring=format)
        print('Created new directory: ', newdir)

    if not base_exists:
        Directory.CreateDirectory(basefolder)
        newdir = basefolder

    return newdir


def createfolder(basedir, formatstring='%Y-%m-%d_%H-%M-%S'):
    """Creates a new folder inside an existing folder using a specific format

    :param basedir: Folder inside which the new directory should be created
    :type basedir: str
    :param formatstring: String format for the new folder, defaults to '%Y-%m-%d_%H-%M-%S'
    :type formatstring: str, optional
    :return: Newly created directory
    :rtype: str
    """

    # construct new directory name based on date and time
    newdir = Path.Combine(basedir, datetime.now().strftime(formatstring))
    # check if the new directory (for whatever reasons) already exists
    try:
        newdir_exists = Directory.Exists(newdir)
        if not newdir_exists:
            # create new directory if is does not exist
            Directory.CreateDirectory(newdir)
        if newdir_exists:
            # raise error if it really already exists
            raise SystemExit
    except OSError as e:
        if e.errno != errno.EEXIST:
            newdir = None
            raise  # This was not a "directory exist" error..

    return newdir


def cloneexp(expname, prefix='GA_', save=True, reloadexp=True):
    """Clone an existing ZenExperiment.

    :param expname: Name of the ZenExperiment
    :type expname: ZenExperiment
```

```
    :param prefix: Prefix for the experiment name, defaults to 'GA_'
    :type prefix: str, optional
    :param save: Option to save the cloned experiment, defaults to True
    :type save: bool, optional
    :param reloadexp: option to reload the cloned experiment afterwards, defaults to True
    :type reloadexp: bool, optional
    :return: exp_reload
    :rtype: ZenExperiment or None
    """

    exp = Zen.Acquisition.Experiments.GetByName(expname)
    exp_newname = prefix + expname

    # save experiment
    if save:
        exp.SaveAs(exp_newname, False)
        print('Saved Temporay Experiment as : ', exp_newname)
        # close the original experiment object
        exp.Close()
        time.sleep(1)

    # reload experiment
    if reloadexp:
        exp_reload = Zen.Acquisition.Experiments.GetByName(exp_newname)
    elif not reloadexp:
        exp_reload = None

    return exp_reload


def runSWAF_special(SWAF_exp,
                    delay=5,
                    searchStrategy='Full',
                    sampling=ZenSoftwareAutofocusSampling.Coarse,
                    relativeRangeIsAutomatic=False,
                    relativeRangeSize=500,
                    timeout=0):
    """Execute SWAF for a ZenExperiment with custom settings.

    :param SWAF_exp: ZenExperiment containing the SWAF settings
    :type SWAF_exp: ZenExperiment
    :param delay: Hardware delay to wait until "everything" has moved into place, defaults to 5
    :type delay: int, optional
    :param searchStrategy: Search Strategy for the SWAF, defaults to 'Full'
    :type searchStrategy: str, optional
    :param sampling: Sampling for the SWAF, defaults to ZenSoftwareAutofocusSampling.Coarse
    :type sampling: ZenSoftwareAutofocusSampling, optional
    :param relativeRangeIsAutomatic: Set SWAF range automatically, defaults to False
    :type relativeRangeIsAutomatic: bool, optional
    :param relativeRangeSize: Relative range of SWAF [micron], defaults to 500
    :type relativeRangeSize: int, optional
    :param timeout: Timeout for SWAF [s], defaults to 0
    :type timeout: int, optional
    :return: zSWAF
    :rtype: float
    """

    # get current z-Position
    zSWAF = Zen.Devices.Focus.ActualPosition
    print('Z-Position before special SWAF :', zSWAF)

    # set DetailScan active and wait for moving hardware due to settings
    SWAF_exp.SetActive()
    time.sleep(delay)

    # set SWAF parameters
    SWAF_exp.SetAutofocusParameters(searchStrategy=searchStrategy,
                                    sampling=sampling,
                                    relativeRangeIsAutomatic=relativeRangeIsAutomatic,
                                    relativeRangeSize=relativeRangeSize)
    try:
        print('Running special SWAF ...')
        zSWAF = Zen.Acquisition.FindAutofocus(SWAF_exp, timeoutSeconds=timeout)
    except ApplicationException as e:
        print('Application Exception : ', e.Message)
    except TimeoutException as e:
        print(e.Message)

    print('Z-Position after initial SWAF [micron]: ', zSWAF)

    return zSWAF


def run_postprocessing(image, parameters={}, func='topography'):
    """Example template to add post-processing functions.

    :param image: Image to be processed
    :type image: ZenImage
    :param parameters: Dictionary with processing parameters for function, defaults to {}
```

```
    :type parameters: dict, optional
    :param func: Function name, defaults to 'topography'
    :type func: str, optional
    :return: image or processed image
    :rtype: ZenImage
    """

    if func == 'topography':

        noise_low = parameters['noise_low']
        noise_high = parameters['noise_high']
        outputfolder = parameters['outputfolder']
        ext = parameters['extension']

        # converting to topo, with defined FZ noisecut
        # in filter settings: 0-255 means no filter, 1-254 means cut one gray scale from top and from bottom
        imgtop = Zen.Processing.Transformation.Topography.CreateTopography(image, noise_low, noise_high)

        # saving file to the directory
        topo_filepath = Path.Combine(outputfolder, Path.GetFileNameWithoutExtension(image.FileName) + ext)
        Zen.Processing.Utilities.ExportHeightmapFromTopography(imgtop, topo_filepath)
        print('Exported to : ', topo_filepath)
        imgtop.Close()

    return image


def apply_recall_focus():
    """Recall the stored Z-Value for Definite Focus 2 (DF2).

    :return: [description]
    :rtype: [type]
    """

    # get the z-position
    zpos = Zen.Devices.Focus.ActualPosition

    try:
        # apply RecallFocus for the current position
        Zen.Acquisition.RecallFocus()
        zpos = Zen.Devices.Focus.ActualPosition
        print('Recall Focus (Definite Focus) applied.')
        print('Updated Z-Position: ', zpos)
    except ApplicationException as e:
        print('Application Exception : ', e.Message)
        print('Recalling Focus (Definite Focus 2) failed.')

    print('New Z-Position before Detail Experiment will start:', zpos)

    return zpos


def modify_tileexperiment_rect(tileexp, blockindex=0,
                               bcwidth=1.0,
                               bcheight=1.0,
                               xpos=1.0,
                               ypos=1.0,
                               zpos=1.0):
    """Modify an existing ZenExperiment with an existing Tileregion by adding
    an new rectangular Tileregion.

    :param tileexp: ZenExperiment to be modified with a rectangular TileRegion.
    :type tileexp: ZenExperiment
    :param blockindex: Experiment Block index, defaults to 0
    :type blockindex: int, optional
    :param bcwidth: Width of the new TileRegion [micron], defaults to 1.0
    :type bcwidth: float, optional
    :param bcheight: Height of the new Tileregion [micron], defaults to 1.0
    :type bcheight: float, optional
    :param xpos: StageX of the new Tileregion center [micron], defaults to 1.0
    :type xpos: float, optional
    :param ypos: StageY of the new TileRegion center [micron], defaults to 1.0
    :type ypos: float, optional
    :param zpos: StageZ of the new TileRegion [micron], defaults to 1.0
    :type zpos: float, optional
    :return: tileexp
    :rtype: ZenExperiment
    """

    print('Width and Height : ', '%.2f' % bcwidth, '%.2f' % bcheight)
    print('Modifying Tile Properties XYZ Position and width and height.')

    # Modify the XYZ position and size of the TileRegion on-the-fly
    print('Starting Z-Position for current Object: ', '%.2f' % zpos)
    print('New Tile Properties: ', '%.2f' % xpos, '%.2f' % ypos, '%.2f' % zpos, '%.2f' % bcwidth, '%.2f' % bcheight)
    tileexp.ClearTileRegionsAndPositions(blockindex)
    try:
        tileexp.AddRectangleTileRegion(blockindex, xpos, ypos, bcwidth, bcheight, zpos)
    except ApplicationException as e:
```

```
        print('Application Exception : ', e.Message)

    return tileexp


def modify_tileexperiment_polygon(tileexp, polyregion,
                                  blockindex=0,
                                  zpos=0.0):
    """Modify an existing ZenExperiment with an existing Tileregion by adding
    an new Polygon Tileregion.

    :param tileexp: ZenExperiment to be modified with a Polygon TileRegion.
    :type tileexp: ZenExperiment
    :param polyregion: List with ZenPoints decribing the polygon in absolute stageXY coordinates [micron]
    :type bcwidth: IList[ZenPoints]
    :param blockindex: Experiment Block index, defaults to 0
    :type blockindex: int, optional
    :param zpos: StageZ of the new TileRegion [micron], defaults to 1.0
    :type zpos: float, optional
    :return: tileexp
    :rtype: ZenExperiment
    """

    print('Modifying Tile Properties using Polygon for ZEN Image Analysis.')
    print('Starting Z-Position for current Object: ', '%.2f' % zpos)
    tileexp.ClearTileRegionsAndPositions(blockindex)
    try:
        tileexp.AddPolygonTileRegion(blockindex, polyregion, zpos)
    except ApplicationException as e:
        print('Application Exception : ', e.Message)

    return tileexp


def create_exppolygon(points, stageTL, scaling):
    """Convert List of pixel-based points from Image Analysis region into
    List[ZenPoints] in absolute stageXY coordinates [micron].

    :param points: List of Points [pixel]
    :type points: List
    :param stageTL: Image Stage Top-Left [micron]
    :type stageTL:
    :param scaling: [description]
    :type scaling: Point
    :return: zplist
    :rtype: List[ZenPoints]
    """

    # create new list of ZenPoints
    pl = []
    zplist= List[ZenPoint]()

    # iterate over all points inside list
    for p in range(len(points)):
        # use StageTopLeft and scaling to convert points to stageXY
        newx = stageTL.X + round(points[p].X * scaling.X, 1)
        newy = stageTL.Y + round(points[p].Y * scaling.Y, 1)
        pl.append(ZenPoint(newx, newy))

    # create IList[ZenPoint]
    zplist.AddRange(pl)

    return zplist


def getclassnames(ias):
    """Get ID and name of classes from Image Analysis Setting

    :param ias: Setting to extract the ID and names from
    :type ias: ZenImageAnalysisSetting
    :return: iaclasses
    :rtype: dict
    """

    # create empty dictionary to conatin the ID and Name
    iaclasses = {}
    classnames = ias.GetRegionClassNames()

    for id in range(0, len(classnames) + 1):
        try:
            cl = ias.GetRegionClass(id)
        except:
            cl = ias.GetRegionsClass(id)

        iaclasses[str(cl.ID)] = cl.Name
        print('ID - ClassName: ', cl.ID, cl.Name)

    return iaclasses
```

```
518
519 ########################################################################
520
521 # clear console output
522 Zen.Application.MacroEditor.ClearMessages()
523
524 # check the location of experiment setups and image analysis settings are stored
525 docfolder = Zen.Application.Environment.GetFolderPath(ZenSpecialFolder.UserDocuments)
526 imgfolder = Zen.Application.Environment.GetFolderPath(ZenSpecialFolder.ImageAutoSave)
527 imgfolder = r'c:\Output\Guided_Acquisition'
528 format = '%Y-%m-%d_%H-%M-%S'
529
530 # get list with all existing experiments and image analysis setup and a short version of that list
531 expfiles = Directory.GetFiles(Path.Combine(docfolder, 'Experiment Setups'), '*.czexp')
532 ipfiles = Directory.GetFiles(Path.Combine(docfolder, 'Image Analysis Settings'), '*.czias')
533 apfiles = Directory.GetFiles(Path.Combine(docfolder, 'APEER Module Settings'), '*.czams')
534 expfiles_short = getshortfiles(expfiles)
535 ipfiles_short = getshortfiles(ipfiles)
536 apfiles_short = getshortfiles(apfiles)
537
538 # Initialize Dialog
539 GuidedAcqDialog = ZenWindow()
540 GuidedAcqDialog.Initialize('Guided Acquisition - Version : ' + str(version))
541 # add components to dialog
542 GuidedAcqDialog.AddLabel('------   Select Overview Experiment   ------')
543 GuidedAcqDialog.AddDropDown('overview_exp', 'Overview Scan Experiment', expfiles_short, 0)
544 GuidedAcqDialog.AddCheckbox('fs_before_overview', 'OPTION - FindSurface (DF only) before Overview', False)
545 GuidedAcqDialog.AddCheckbox('SWAF_before_overview', 'OPTION - SWAF before Overview', False)
546 GuidedAcqDialog.AddIntegerRange('SWAF_ov_initial_range', 'Initial SWAF Range before Overview [micron]', 200, 50, 3000)
547 GuidedAcqDialog.AddLabel('------   Select Image Analysis to detect objects   ------')
548 GuidedAcqDialog.AddDropDown('ip_pipe', 'Image Analysis Pipeline', ipfiles_short, 0)
549 GuidedAcqDialog.AddCheckbox('use_poly', 'Use Polygons from ZEN IA instead of BBox', True)
550 GuidedAcqDialog.AddCheckbox('use_apeer_for_IA', 'Use APEER module run run IA instead', False)
551 GuidedAcqDialog.AddDropDown('ap_pipe', 'APEER Module Settings', apfiles_short, 0)
552 GuidedAcqDialog.AddLabel('------   Select DetailScan Experiment   ------')
553 GuidedAcqDialog.AddDropDown('detailed_exp', 'Detailed Scan Experiment', expfiles_short, 1)
554 GuidedAcqDialog.AddCheckbox('fs_before_detail', 'OPTION - FindSurface (DF only) before Detail', False)
555 GuidedAcqDialog.AddCheckbox('SWAF_before_detail', 'OPTION - SWAF before Detail', False)
556 GuidedAcqDialog.AddIntegerRange('SWAF_detail_initial_range', 'Initial SWAF Range before Detail [micron]', 100, 10, 1000)
557 GuidedAcqDialog.AddCheckbox('recallfocus_beforeDT', 'OPTION - Use RecallFocus (DF only) before Detail', False)
558 GuidedAcqDialog.AddLabel('------   Specify basefolder to save the images   ------')
559 GuidedAcqDialog.AddFolderBrowser('outfolder', 'Basefolder for Images and Data Tables', imgfolder)
560
561 # show the window
562 result = GuidedAcqDialog.Show()
563 if result.HasCanceled:
564     message = 'Macro was canceled by user.'
565     print(message)
566     raise SystemExit
567
568 # get the values and store them
569 OverViewExpName = str(result.GetValue('overview_exp'))
570 ImageAS = str(result.GetValue('ip_pipe'))
571 ApeerMS = str(result.GetValue('ap_pipe'))
572 DetailExpName = str(result.GetValue('detailed_exp'))
573 OutputFolder = str(result.GetValue('outfolder'))
574 fs_beforeOV = result.GetValue('fs_before_overview')
575 SWAF_beforeOV = result.GetValue('SWAF_before_overview')
576 SWAF_beforeOV_range = result.GetValue('SWAF_ov_initial_range')
577 fs_beforeDT = result.GetValue('fs_before_detail')
578 SWAF_beforeDT = result.GetValue('SWAF_before_detail')
579 SWAF_beforeDT_range = result.GetValue('SWAF_detail_initial_range')
580 RecallFocus = result.GetValue('recallfocus_beforeDT')
581 use_apeer = result.GetValue('use_apeer_for_IA')
582 use_polygon = result.GetValue('use_poly')
583
584
585 # print values
586 print('Overview Scan Experiment : ', OverViewExpName)
587 if not use_apeer:
588     print('Image Analysis Pipeline : ', ImageAS)
589     print('Use IA Polygon instead of BBox : ', use_polygon)
590 if use_apeer:
591     print('Use Apeer Module Setting : ', ApeerMS)
592 print('Detailed Scan Experiment : ', DetailExpName)
593 print('Output Folder for Data : ', OutputFolder)
594 print('\n')
595
596 # check directory
597 OutputFolder = dircheck(OutputFolder)
598
599 # create a duplicate of the OVScan experiment to work with
600 OVScan_reloaded = cloneexp(OverViewExpName)
601
602 # active the temporary experiment to trigger its validation
603 OVScan_reloaded.SetActive()
604 time.sleep(hwdelay)
605
606 # check if the experiment contains tile regions
```

```
OVScanIsTileExp = OVScan_reloaded.IsTilesExperiment(blockindex)

############# START OVERVIEW SCAN EXPERIMENT #################

if fs_beforeOV:
    # initial focussing via FindSurface to assure a good starting position
    Zen.Acquisition.FindSurface()
    print('Z-Position after FindSurface: ', Zen.Devices.Focus.ActualPosition)

if SWAF_beforeOV:
    zSWAF = runSWAF_special(OVScan_reloaded,
                            delay=hwdelay,
                            searchStrategy='Full',
                            sampling=ZenSoftwareAutofocusSampling.Coarse,
                            relativeRangeIsAutomatic=False,
                            relativeRangeSize=SWAF_beforeOV_range,
                            timeout=1)

# get the resulting z-position
znew = Zen.Devices.Focus.ActualPosition

# adapt the Overview Scan Tile Experiment with new Z-Position
if OVScanIsTileExp:
    OVScan_reloaded.ModifyTileRegionsZ(blockindex, znew)
    print('Adapted Z-Position of Tile OverView. New Z = ', '%.2f' % znew)

# execute the experiment
print('\nRunning OverviewScan Experiment.\n')
#output_OVScan = Zen.Acquisition.Execute(OVScan_reloaded)
# For testing purposes - Load overview scan image automatically instead of executing the "real" experiment
ovtestimage = r'C:\Users\m1srh\OneDrive - Carl Zeiss AG\Smart_Microscopy_Workshop\datasets\brainslide\OverViewScan.czi'
output_OVScan = Zen.Application.LoadImage(ovtestimage, False)

# the the stage top-left and the scaling of the overview image
stageTL = output_OVScan.GetPositionLeftTop()
ovscaling = output_OVScan.Scaling

# show the overview scan inside the document area
Zen.Application.Documents.Add(output_OVScan)
ovdoc = Zen.Application.Documents.GetByName(output_OVScan.Name)

# save the overview scan image inside the select folder
savepath_ovscan = Path.Combine(OutputFolder, ovscan_name)
output_OVScan.Save(savepath_ovscan)

# get the actual focus value for the overscan independent from the z-value
# before the start of the overview scan using the image metadata
zvalue_ovscan = output_OVScan.Metadata.FocusPositionMicron

############# END OVERVIEW SCAN EXPERIMENT ##################

# run normal ZEN image analysis
if not use_apeer:

    # Load analysis setting created by the wizard or an separate macro
    ias = ZenImageAnalysisSetting()

    # for simulation use: 000 - RareEventExample.czias
    ias.Load(ImageAS)

    # Analyse the image
    Zen.Analyzing.Analyze(output_OVScan, ias)

    # get classes and derive regions names from that
    iasclasses = getclassnames(ias)

    if use_polygon:

        # get the individual image analysis regions from 1st IA class (!!!)

        # ATTENTION: the first single objects class will be used. It has ID = 2
        regions = Zen.Analyzing.GetRegions(output_OVScan, iasclasses['2'])
        print('RegionClassNames: ', iasclasses)
        print('Analysis found ' + str(regions.Count) + 'regions!')

        # create dictionary for polygon regions
        polyregions = {}

        # loop over all regions and get the points of polygon (outline of the object)
        for i in range(0, regions.Count):
            points = regions[i].GetPolygon()

            # convert points to stage coordinates
            polyregions[i] = create_exppolygon(points, stageTL, ovscaling)

    # Create Zen table with results for all detected objects (parent class)
    AllObj = Zen.Analyzing.CreateRegionsTable(output_OVScan)

    # Create Zen table with results for each single object
```

```
    SingleObj = Zen.Analyzing.CreateRegionTable(output_OVScan)

    # check for existence of required column names inside table
    soi = SingleObj.GetBoundsColumnInfoFromImageAnalysis(True)

    # 1st item is a bool indicating if all required columns could be found
    columnsOK = soi.AreRequiredColumnsAvailabe

    if not columnsOK:
        print('Execution stopped. Required Columns are missing.')
        raise Exception('Execution stopped. Required Columns are missing.')

    # show and save data tables to the specified folder
    Zen.Application.Documents.Add(AllObj)
    Zen.Application.Documents.Add(SingleObj)
    AllObj.Save(Path.Combine(OutputFolder, 'OverviewTable.csv'))
    SingleObj.Save(Path.Combine(OutputFolder, 'SingleObjectsTable.csv'))

# use APEER module to detect the objects
if use_apeer:

    # read it from settings file
    ams = ZenApeer.Onsite.ModuleSetting()

    # loaf the APEER module seeting b - remove *.czams file extension first
    ams.Load(Path.GetFileNameWithoutExtension(ApeerMS))

    print('-----   Apeer Module Setting   -----')
    print('Module Name    : ', ams.ModuleName)
    print('Module Version : ', ams.ModuleVersion)
    print('Module Parameters    : ', ams.Parameters)

    # get module and check
    mymodule, version_found = get_module(ams.ModuleName, module_version=ams.ModuleVersion)

    # exit if the check failed
    if mymodule is None or not version_found:
        print('Module check failed. Exiting.')
        raise SystemExit

    # get the module parameters for the specified module
    module_params = ZenApeer.Onsite.GetSampleModuleParameters(ams.ModuleName, ams.ModuleVersion)

    # get the module input programmatically
    module_inputs = get_module_inputs(module_params)

    # create the required dictionary with the correct key and value
    input_image = {module_inputs[0]: savepath_ovscan}

    # create the path to save the results
    #savepath_apeer = Path.Combine(OutputFolder, 'apeer_results')
    savepath_apeer = OutputFolder

    # create the output directory if not existing already
    if not Directory.Exists(savepath_apeer):
        Directory.CreateDirectory(savepath_apeer)

    # run the local APEER module with using keywords
    try:
        runoutputs, status, log = ZenApeer.Onsite.RunModule(moduleName=ams.ModuleName,
                                                            moduleVersion=ams.ModuleVersion,
                                                            inputs=input_image,
                                                            parameters=ams.Parameters,
                                                            storagePath=savepath_apeer)

        for op in runoutputs.GetEnumerator():
            print('-----    Outputs   -----')
            print(op.Key, ' : ', op.Value)

    except ApplicationException as e:
        print('Module Run failded.', e.Message)
        raise SystemExit

    # get results storage locations
    # IMPORTANT: To be used inside Guided Acquisition
    # the APEER Module is expected to have
    # at least those to outputs with exactly those names

    if not runoutputs.ContainsKey('segmented_image'):
        print('No output : segmented_image')
        raise SystemExit

    if not runoutputs.ContainsKey('objects_table'):
        print('No output : objects_table')
        raise SystemExit

    # load the segmented image and make sure the pyramid is calculated
    segmented_image = Zen.Application.LoadImage(runoutputs['segmented_image'], False)
    Zen.Processing.Utilities.GenerateImagePyramid(segmented_image, ZenBackgroundMode.Black)
```

```
    Zen.Application.Documents.Add(segmented_image)

    # auto-display min-max
    ids = segmented_image.DisplaySetting.GetAllChannelIds()
    for id in ids:
        segmented_image.DisplaySetting.SetParameter(id, 'IsAutoApplyEnabled', True)

    # initialize ZenTable object and load CSV file
    SingleObj = ZenTable()
    SingleObj.Load(runoutputs['objects_table'])
    Zen.Application.Documents.Add(SingleObj)

    # get all columns as dict with columnIDs
    colID = get_columns(SingleObj)

    # define the required column names here
    col2check = ('bbox_center_stageX', 'bbox_center_stageY', 'bbox_width_scaled', 'bbox_height_scaled')

    # check if all columns exist
    if all(key in colID for key in col2check):
        print('All required columns found.')
    else:
        print('Not All required columns found. Exiting.')
        raise SystemExit

# check the number of detected objects = rows inside image analysis table
num_POI = SingleObj.RowCount

############## Prepare DetailScan #################

print('Starting DetailScan ...')

if not use_apeer:
    xpos_1st = SingleObj.GetValue(0, soi.CenterXColumnIndex)
    ypos_1st = SingleObj.GetValue(0, soi.CenterYColumnIndex)

if use_apeer:
    xpos_1st = SingleObj.GetValue(0, colID['bbox_center_stageX'])
    ypos_1st = SingleObj.GetValue(0, colID['bbox_center_stageY'])

# move to 1st detected object to be at a XY-position that makes sense
Zen.Devices.Stage.MoveTo(xpos_1st + dx_detector, ypos_1st + dy_detector)

# and move the the z-values from the overview scan image
Zen.Devices.Focus.MoveTo(zvalue_ovscan)

# create an duplicate of the DetailScan experiment to work with
DetailScan_reloaded = cloneexp(DetailExpName)

# active the temporary experiment to trigger its validation
DetailScan_reloaded.SetActive()
time.sleep(hwdelay)
# check if the experiment contains tile regions
DetailIsTileExp = DetailScan_reloaded.IsTilesExperiment(blockindex)

# test snap to change to the valid settings, e.g. the objective from the DetailScan
testsnap = Zen.Acquisition.AcquireImage(DetailScan_reloaded)
print('Acquire Test Snap using setting from DetailScan')
testsnap.Close()
# wait for moving hardware due to settings
time.sleep(hwdelay)

if fs_beforeDT:
    try:
        # initial focussing via FindSurface to assure a good starting position
        Zen.Acquisition.FindSurface()
        print('Z-Position after FindSurface: ', Zen.Devices.Focus.ActualPosition)
    except ApplicationException as e:
        print('Application Exception : ', e.Message)
        print('FindSurface (Definite Focus) failed.')

if SWAF_beforeDT:
    zSWAF = runSWAF_special(DetailScan_reloaded,
                            delay=hwdelay,
                            searchStrategy='Full',
                            sampling=ZenSoftwareAutofocusSampling.Coarse,
                            relativeRangeIsAutomatic=False,
                            relativeRangeSize=SWAF_beforeDT_range,
                            timeout=0)

userecallfocus = False

if RecallFocus:
    try:
        # store current focus position inside DF to use it with RecallFocus
        Zen.Acquisition.StoreFocus()
        userecallfocus = True
    except ApplicationException as e:
        print('Application Exception : ', e.Message)
```

```
        print('StoreFocus (Definite Focus) failed.')
        userecallfocus = False


############# START DETAILED SCAN EXPERIMENT #############

# again get the resulting focus position
zpos = Zen.Devices.Focus.ActualPosition

if not use_apeer:

    # check for the column 'ID' which is required
    ID_exists, column_ID = checktableentry(SingleObj, entry2check='ID')

    # execute detailed experiment at the position of every detected object
    for obj in range(SingleObj.RowCount):

        # get the object information from the position table
        POI_ID = SingleObj.GetValue(obj, column_ID)

        # get XY-stage position from table
        xpos = SingleObj.GetValue(obj, soi.CenterXColumnIndex)
        ypos = SingleObj.GetValue(obj, soi.CenterYColumnIndex)

        # move to the current position
        Zen.Devices.Stage.MoveTo(xpos + dx_detector, ypos + dy_detector)
        print('Moving Stage to Object ID:', POI_ID, ' at :', '%.2f' % xpos, '%.2f' % ypos)

        # try to apply RecallFocus (DF only) when this option is used
        if userecallfocus:
            zpos = apply_recall_focus()

        # if DetailScan is a Tile Experiment
        if DetailIsTileExp:

            print('Detailed Experiment contains TileRegions.')
            # Modify tile center position - get bounding rectangle width and height in microns
            bcwidth = SingleObj.GetValue(obj, soi.WidthColumnIndex)
            bcheight = SingleObj.GetValue(obj, soi.HeightColumnIndex)

            if use_polygon:
                # modify the experiment with th respective polygon region
                DetailScan_reloaded = modify_tileexperiment_polygon(DetailScan_reloaded, polyregions[obj],
                                                                    blockindex=0,
                                                                    zpos=zpos)
            else:
                # modify the experiment
                DetailScan_reloaded = modify_tileexperiment_rect(DetailScan_reloaded,
                                                                 blockindex=blockindex,
                                                                 bcwidth=bcwidth,
                                                                 bcheight=bcheight,
                                                                 xpos=xpos,
                                                                 ypos=ypos,
                                                                 zpos=zpos)

        if not DetailIsTileExp:
            print('Detailed Experiment does not contains TileRegions. Nothing to modify.')

        # execute the DetailScan experiment
        print('Running Detail Scan Experiment at new XYZ position.')
        try:
            output_detailscan = Zen.Acquisition.Execute(DetailScan_reloaded)
        except ApplicationException as e:
            print('Application Exception : ', e.Message)

        # get the image data name
        dtscan_name = output_detailscan.Name

        """
        Modification for multiscene images: container name needs to be part
        of the filename of the detailed Scan.
        First check if Column "Image Scene Container Name" or "WellId" is available
        and then add Container Name to filename.
        """

        wellid_exist, column_wellid = checktableentry(SingleObj, entry2check='ImageSceneContainerName')

        # in case Image Scene Container Name is not defined
        if not wellid_exist:
            message = 'Missing column ImageSceneContainerName in Image Analysis Results.\nPlease Select Features inside the
            ↪  Image Analysis when needed.'
            print(message)
            container_name = 'empty'

        # save the image data to the selected folder and close the image
        output_detailscan.Save(Path.Combine(OutputFolder, output_detailscan.Name))
        output_detailscan.Close()

        # rename the CZI regarding to the object ID - Attention - IDs start with 2 !!!
```

```
        if not wellid_exist:
            # no wellID found inside table
            newname_dtscan = 'DTScan_ID_' + str(POI_ID) + '.czi'
        if wellid_exist:
            # wellID was found inside table
            well_id = SingleObj.GetValue(obj, column_wellid)
            newname_dtscan = 'DTScan_Well_' + str(well_id) + '_ID_' + str(POI_ID) + '.czi'

        print('Renaming File: ' + dtscan_name + ' to: ' + newname_dtscan + '\n')
        File.Move(Path.Combine(OutputFolder, dtscan_name), Path.Combine(OutputFolder, newname_dtscan))

        ############ OPTIONAL POSTPROCESSING ###############

        if do_postprocess:

            # do the postprocessing
            image2process = Zen.Application.LoadImage(newname_dtscan, False)

            # define the parameters for processing: Topography Export
            parameters = {}
            parameters['noise_low'] = 1
            parameters['noise_high'] = 254
            parameters['outputfolder'] = OutputFolder
            parameters['extension'] = '.sur'

            # run the processing and export and close the image
            image2process = run_postprocessing(image2process, parameters=parameters, func='topography')
            image2process.Close()

if use_apeer:

    # execute detailed experiment at the position of every detected object
    for obj in range(SingleObj.RowCount):

        # get the object information from the position table
        POI_ID = obj

        # get XY-stage position from table
        xpos = SingleObj.GetValue(obj, colID['bbox_center_stageX'])
        ypos = SingleObj.GetValue(obj, colID['bbox_center_stageY'])

        # move to the current position
        Zen.Devices.Stage.MoveTo(xpos + dx_detector, ypos + dy_detector)
        print('Moving Stage to Object:', POI_ID + 1, ' at :', '%.2f' % xpos, '%.2f' % ypos)

        # try to apply RecallFocus (DF only) when this option is used
        if userecallfocus:
            zpos = apply_recall_focus()

        # if DetailScan is a Tile Experiment
        if DetailIsTileExp:


            # Modify tile center position - get bounding rectangle width and height [microns]
            print('Detailed Experiment contains TileRegions.')
            bcwidth = SingleObj.GetValue(obj, colID['bbox_width_scaled'])
            bcheight = SingleObj.GetValue(obj, colID['bbox_height_scaled'])

            # modify the ZenExperiment for the DetailScan
            DetailScan_reloaded = modify_tileexperiment_rect(DetailScan_reloaded,
                                                             blockindex=blockindex,
                                                             bcwidth=bcwidth,
                                                             bcheight=bcheight,
                                                             xpos=xpos,
                                                             ypos=ypos,
                                                             zpos=zpos)

        if not DetailIsTileExp:
            print('Detailed Experiment does not contains TileRegions. Nothing to modify.')

        # execute the DetailScan experiment
        print('Running Detail Scan Experiment at new XYZ position.')
        try:
            output_detailscan = Zen.Acquisition.Execute(DetailScan_reloaded)
        except ApplicationException as e:
            print('Application Exception : ', e.Message)

        # get the image data name
        dtscan_name = output_detailscan.Name

        """
        Modification for multiscene images: container name needs to be part
        of the filename of the detailed Scan.
        First check if Column "Image Scene Container Name" or "WellId" is available
        and then add Container Name to filename.
        """

        if 'WellId' in colID:
            print('WellId column found.')
```

```
            wellid_exist = True
            column_wellid = colID['WellId']
        else:
            print('WellId column not found.')
            wellid_exist = False
            column_wellid = None

        # in case Image Scene Container Name is not defined
        if not wellid_exist:
            message = 'Missing column WellId in DataTable.\nPlease modify your image analysis inside the APEER module or manually add column.'
            print(message)
            container_name = 'empty'

        # save the image data to the selected folder and close the image
        output_detailscan.Save(Path.Combine(OutputFolder, output_detailscan.Name))
        output_detailscan.Close()

        # rename the CZI regarding to the object ID - Attention - IDs start with 2 !!!
        if not wellid_exist:
            # no wellID found inside table
            newname_dtscan = 'DTScan_ID_' + str(POI_ID) + '.czi'
        if wellid_exist:
            # wellID was found inside table
            well_id = SingleObj.GetValue(obj, column_wellid)
            newname_dtscan = 'DTScan_Well_' + str(well_id) + '_ID_' + str(POI_ID) + '.czi'

        print('Renaming File: ' + dtscan_name + ' to: ' + newname_dtscan + '\n')
        File.Move(Path.Combine(OutputFolder, dtscan_name), Path.Combine(OutputFolder, newname_dtscan))

        ############ OPTIONAL POSTPROCESSING ###############

        if do_postprocess:

            # do the postprocessing
            image2process = Zen.Application.LoadImage(newname_dtscan, False)

            # define the parameters for processing: Topography Export
            parameters = {}
            parameters['noise_low'] = 1
            parameters['noise_high'] = 254
            parameters['outputfolder'] = OutputFolder
            parameters['extension'] = '.sur'

            # run the processing and export and close the image
            image2process = run_postprocessing(image2process, parameters=parameters, func='topography')
            image2process.Close()


############# END DETAILED SCAN EXPERIMENT #############

# restore the original OVScan experiment
OVScan_orig = Zen.Acquisition.Experiments.GetByName(OverViewExpName)
OVScan_orig.SetActive()

# delete the temporay experiments when all loops are finished
Zen.Acquisition.Experiments.Delete(OVScan_reloaded)
Zen.Acquisition.Experiments.Delete(DetailScan_reloaded)

# show the overview scan document again at the end
Zen.Application.Documents.ActiveDocument = ovdoc
print('All Positions done. Guided Acquisition Workflow finished.')
```

9 ToDo's and Limitations

The current version of this tool has still limitations and room for improvement.

- No yet built-in check to avoid double detailed scans for objects close to each other within one field of view.

10 Disclaimer

DISCLAIMER:

The script and application are free and can be used by everybody on their own risk.

Especially be aware of the fact that automated stage movements might damage hardware if the system is not setup properly and not correctly calibrated.

Please check everything in simulation mode first!

Carl Zeiss Microscopy GmbH's ZEN software allows connection to the third party software, Python. Therefore Carl Zeiss Microscopy GmbH undertakes no warranty concerning Python, makes no representation that Python will work on your hardware, and will not be liable for any damages caused by the use of this extension. By running and using this script confirm that you understood the involved risks you agree to this disclaimer.