Legacy code overview

mesoSPIM Control software

Author: Fabian Voigt

Currently, everything is in one big file to avoid the messiness of relative imports

TODO: CHANGE

class `mesoSPIM.legacy.mesoSPIM.mesoSPIM_control.`**`Script_Repository`** [source]

Bases: **`PyQt5.QtCore.QObject`**

Class that contains scripts.

Things should be written in here using the mutex.

class `mesoSPIM.legacy.mesoSPIM.mesoSPIM_control.`**`mesoSPIM_Camera`** [source]

Bases: **`PyQt5.QtCore.QObject`**

Class for the Hamamatsu Camera

Will run in its own thread

`add_images_to_stack`() [source]

`end_stack`() [source]

`frame`

`live`() [source]

camera running in live mode

`prepare_stack`() [source]

`set_exposure_time`() [source]

`set_line_interval`() [source]

`status`

`stop`() [source]

class `mesoSPIM.legacy.mesoSPIM.mesoSPIM_control.`**`mesoSPIM_Core`**(*config*) [source]

Bases: **`PyQt5.QtCore.QObject`**

The mesoSPIM core takes care of all interactions with NI hardware and is thus the pacemaker of the microscope (responsible for synchronizing all digital and analog outs).

TODO: In the future, the core will take care of script execution as well.

`bundle_galvo_and_etl_waveforms`() [source]

Stacks the Galvo and ETL waveforms into a numpy array adequate for the NI cards.

In here, the assignment of output channels of the Galvo / ETL card to the corresponding output channel is hardcoded: This could be improved.

`calculate_number_of_stack_planes`(*start_z*, *end_z*, *stepsize*) [source]

Returns the number of planes

Uses int( ) to round down to the nearest integer.

`close_image_series`() [source]

Cleans up after series without waveform update

`close_shutters`() [source]

`close_tasks`() [source]

Closes the tasks for triggering, analog and counter outputs.

Tasks should only be closed are they are stopped.

`create_filename`() [source]

`create_tasks`() [source]

Creates a total of four tasks for the mesoSPIM:

These are: - the master trigger task, a digital out task that only provides a trigger pulse for the others - the camera trigger task, a counter task that triggers the camera in lightsheet mode - the galvo task (analog out) that controls the left & right galvos for creation of

the light-sheet and shadow avoidance

- the ETL & Laser task (analog out) that controls all the laser intensities (Laser should only be on when the camera is acquiring) and the left/right ETL waveforms

`execute_script`() [source]

Execute script

The script QMutex is locked to avoid the script being changed while there is an execution happening.

Using the traceback module to provide decent exception messages when script execution fails.

`lightsheet_alignment_mode`() [source]

Switches shutters after each image to allow coalignment of both lightsheets

`live`() [source]

`move_absolute`(*dict*, *wait_until_done=False*) [source]

Helper method to allow running movements with or without waiting for completion

`move_relative`(*dict*, *wait_until_done=False*) [source]

Helper method to allow running movements with or without waiting for completion

`open_shutters`() [source]

Here, the possible values are hardcoded which is a DRY violation

`prepare_image_series`() [source]

Prepares an image series without waveform update

`run_stack`(*start_z*, *end_z*, *stepsize*, *return_to_start=False*) [source]

Runs a stack

`run_tasks`() [source]

Runs the tasks for triggering, analog and counter outputs

Firstly, the master trigger triggers all other task via a shared trigger line (PFI line as given in the config file).

For this to work, all analog output and counter tasks have to be started so that they are waiting for the trigger signal.

`save_etl_parameters_to_csv`() [source]

Saves the current ETL left/right offsets and amplitudes from the values to the ETL csv files

The .csv file needs to contain the following columns:

Wavelength Zoom ETL-Left-Offset ETL-Left-Amp ETL-Right-Offset ETL-Right-Amp

Creates a temporary cfg file with the ending _tmp

set_etl_l_waveform(*amplitude*, *offset*) [source]

Allows simple external changing of the ETL left parameters

set_etl_r_waveform(*amplitude*, *offset*) [source]

Allows simple external changing of the ETL right parameters

set_galvo_waveforms(*freq=99.5*, *amp=3*, *offset_l=0*, *offset_r=0*, *phase_l=1.5707963267948966*, *phase_r=1.5707963267948966*) [source]

Updates galvo waveforms, assumes that the frequencies and amplitudes are the same on both sides

set_intensity(*intensity*) [source]

Updates Intensity of the activated laser

set_laser(*laser*, *intensity*) [source]

Enables laser line, sets laser intensity, updates laser and ETL waveforms with values from the parameter table

laser: String intensity: Float between 0 and 100 %

set_laser_intensity(*laserline*, *intensity*, *delay_percent=10*, *pulse_percent=80*) [source]

set_shutter_selection(*shutterstring*) [source]

set_sweeptime(*sweeptime*) [source]

Sets the sweeptime in seconds and updates all related waveforms

Things to update: ETL and Galvo parameters are retained, but laser intensity etc. has to be updated

set_zoom(*zoom*) [source]

Little helper method: Because the mesoSPIM core is not handling the serial Zoom connection.

snap() [source]

snap_image() [source]

Snaps a single image after updating the waveforms.

Can be used in acquisitions where changing waveforms are required, but there is additional overhead due to the need to write the waveforms into the buffers of the NI cards.

snap_image_in_series() [source]

Snaps and image from a series without waveform update

stack() [source]

Stack algorithm

Get start and end points & stepsize from the state Go to start position Update waveforms Lock waveform controls Open shutters for i in range(…):

> Snap image in series Move stage Signal progress How to escape the loop?
>
> > Break when stopflag pops up

Close shutters (no shutterswitch in this one) Go back to start position Enable waveform controls

start_tasks() [source]

Starts the tasks for camera triggering and analog outputs

If the tasks are configured to be triggered, they won't output any signals until run_tasks() is called.

stop() [source]

Signal-invoked local stop flag telling all listeners to stop.

TODO: Investigate whether this might be better handled with an internal signal.

stop_tasks() [source]

Stops the tasks for triggering, analog and counter outputs

update_etl() [source]

Little helper method to update ETL parameters

update_etl_parameters_from_csv(*cfg_path*, *laser*, *zoom*) [source]

Updates the internal ETL left/right offsets and amplitudes from the values in the ETL csv files

The .csv file needs to contain the follwing columns:

Wavelength Zoom ETL-Left-Offset ETL-Left-Amp ETL-Right-Offset ETL-Right-Amp

update_position_state(*position_dict*) [source]

update_stack_parameters() [source]

Updates the stack parameters: calculates z_planes after z_start, z_end, z_stepsize are known

visual_mode() [source]

Provides a simple way to show brains visually with a stereomicroscope

TODO: Sweeptime-induced laser shuttering is still there. Not having this would mean we need a continous output task for the Galvos, which would be a bit of work.

write_line(*file*, *key=''*, *value=''*) [source]

Little helper method to write a single line with a key and value for metadata

Adds a line break at the end.

write_metadata(*path*) [source]

Writes a metadata.txt file

Path contains the file to be written Filename

write_waveforms_to_tasks() [source]

Write the waveforms to the slave tasks

class mesoSPIM.legacy.mesoSPIM.mesoSPIM_control.**mesoSPIM_GUI** [source]

Bases: **PyQt5.QtWidgets.QMainWindow**

Main application window which instantiates a worker object and moves it to a thread.

button1() [source]

select the first laser

button10() [source]

button11() [source]

Increment filter

button12() [source]

Decrement filter

button13() [source]

Increment zoom

button14() [source]

Decrement zoom

button15() [source]

Zero XYZ

button16() [source]

Zero F

button17() [source]

Select left shutter

button18() [source]

Select left shutter

button19() [source]

Select left shutter

button2() [source]

button20() [source]

Select right shutter

button21() [source]

button22() [source]

button23() [source]

button24() [source]

button25() [source]

button26() [source]

Increase laser intensity

The LaserSliderChangeCount is used to decrease the number of events, otherwise the slider would be moved too quickly.

button27() [source]

Decrease laser intensity

The LaserSliderChangeCount is used to decrease the number of events, otherwise the slider would be moved too quickly.

button28() [source]

button29() [source]

button3() [source]

button4() [source]

button5() [source]

button6() [source]

button7() [source]

button8() [source]

button9() [source]

button_handler(*string*, *index*, *message*) [source]

button_handler_method(*string*, *index*, *method_to_call*) [source]

choose_etl_config() [source]

File dialog for choosing the config file

TODO: Check that this is really a .csv-File

disable_alignment_mode_buttons() [source]

disable_controls() [source]

`disable_mode_control_buttons`() [source]

If the microscope is in any acquisition state, it should disable ALL the buttons that can cause it to go into another mode and enable the stop button.

ONLY the stop button or proper finishing of the acquisition allow the GUI to return to the original state.

`display_status_message`(*string*, *time=0*) [source]

Displays a message in the status bar for a time in ms

If time=0, the message will stay.

`enable_alignment_mode_buttons`() [source]

`enable_controls`() [source]

`enable_stop_button`() [source]

`execute_script`() [source]

If the script comes as a string, we can just exec it

`farm_panel_handler`(*data*) [source]

Buttons 1 to 8

`get_bin`(*x*, *n=0*) [source]

Get the binary representation of x.

Parameters:
- **x** (*int*) –
- **n** (*int*) – Minimum number of digits. If x needs less digits in binary, the rest is filled with zeros.

Returns:

Return type: str

`joystick_handler`(*axis*, *value*) [source]

`joystick_persistent_update`() [source]

`joystick_timer_start`() [source]

The interval is hardcoded and was found to be ok with PI stages

`joystick_timer_stop`() [source]

`lightsheet_alignment_mode`() [source]

Run in light-sheet switching mode: interleaved left/right shuttering for coalignment of the light-sheets

TODO: When Stack / Script buttons become available, disable them as well

`live`() [source]

Enter live mode

`load_sample`() [source]

`load_script`() [source]

Load a script

The empty string in the method arguments ensures that it remembers the last location from where a file was opened.

`mark_stack_end`() [source]

`mark_stack_start`() [source]

`pos2str`(*position*) [source]

Little helper method for converting positions to strings

`relative_movement`(*dict*) [source]

Takes a single-axis movement dict in the form

{'z_rel': 100} and sends it out as {'x_rel': 0,'y_rel': 0, 'z_rel': 100, 'f_rel': 0, 'theta_rel': 0}

`reset_GUI`() [source]

Resets the GUI after any acquisition (live, stack, alignment etc ran)

TODO: When Stack / Script buttons become available, reset them as well

`sample_handler`(*data*) [source]

`save_etl_config`() [source]

Save current ETL parameters into config

`save_script`() [source]

Save a script

`set_etl_increments`() [source]

`set_etl_parameters`() [source]

`set_exposure_time`() [source]

Display is in ms, value needed is in seconds

`set_filter`() [source]

`set_galvo_parameters`() [source]

Update Galvo Parameters from the values in the parameter tab

`set_intensity`() [source]

Updates laser intensity of the running laser

`set_laser_and_intensity`() [source]

Sends laser and laser intensity in %

`set_laser_via_button`(*index*, *laser*) [source]

`set_line_interval`() [source]

Change line interval

`set_progressbar_to_busy`() [source]

If min and max of a progress bar are 0, it shows a "busy" indicator

`set_progressbar_to_standard`() [source]

`set_shutter_selection`() [source]

`set_sweeptime`() [source]

Display is in ms, value needed is in seconds

TODO: Functionality needs to be added

`set_tiling_start_position`() [source]

`set_tiling_z_end_position`() [source]

`set_z_stepsize`() [source]

`set_zoom`() [source]

`snap`() [source]

Snap a single image

`stack`() [source]

stop() [source]

Should stop everything

stop_movement() [source]

tiling() [source]

unload_sample() [source]

update_etl_parameters() [source]

update_file_prefix() [source]

update_folder() [source]

Get a new folder name for saving the acquisition

update_position_indicators(*dict*) [source]

update_progressbar(*value*) [source]

update_start_number() [source]

update_tiling_parameters() [source]

Here, the tiling parameters are calculated and set by the GUI

These are: - number of z_planes per substack - total number of planes

update_z_planes() [source]

visual_mode() [source]

Run in visual mode without ETL parameters

TODO: Disable ETL controls and reenable them later as part of the stop method ?

zeroLeftETL() [source]

Zeros the amplitude of the left ETL for faster alignment

zeroRightETL() [source]

Zeros the amplitude of the right ETL for faster alignment

zero_focus() [source]

zero_rot() [source]

zero_xy() [source]

zero_xyz() [source]

zero_z() [source]

class mesoSPIM.legacy.mesoSPIM.mesoSPIM_control.**mesoSPIM_Global_Signals** [source]

Bases: **PyQt5.QtCore.QObject**

Contains general state and config of the microscope

acquisition

add_images_to_stack

end_stack

execute_script

finished

joystick_timer_start

joystick_timer_stop

lightsheet_alignment_mode

live

load_sample

move_absolute

move_absolute_and_wait_until_done

move_relative

move_relative_and_wait_until_done

prepare_stack

progress

save_etl_parameters_to_csv

set_filter

set_intensity

set_laser_and_intensity

set_zoom

snap

stack

status_message

stop

stop_movement

unload_sample

update_camera_exposure

update_camera_line_interval

update_etl_from_csv

update_etl_gui

update_gui

update_position

update_stack_parameters

update_sweeptime

update_waveforms

visual_mode

zero_focus

zero_rot

zero_xy

zero_xyz

zero_z

class `mesoSPIM.legacy.mesoSPIM.mesoSPIM_control.`**`mesoSPIM_Serial`**(*config*) [source]

Bases: **`PyQt5.QtCore.QObject`**

The mesoSPIM Serial class takes care of all interactions via serial ports.

These are:

- Stages (translation and rotation)
- Zoom Control

- Filter wheels

The reason for running this in its own thread is that stages require frequent polling of coordinates.

`block_till_controller_is_ready`() [source]

Blocks further execution (especially during referencing moves) till the PI controller returns ready

`check_target_safety`(*x*, *y*, *z*, *f*, *theta*) [source]

`create_internal_position_dict`() [source]

`create_position_dict`() [source]

`load_sample`() [source]

`move_absolute`(*dict*) [source]

PI move absolute method

Lots of implementation details in here, should be replaced by a facade

TODO: Also lots of repeating code. TODO: DRY principle violated TODO: Emission of a stop signal a good idea or not?

`move_relative`(*dict*) [source]

`move_relative_and_wait`(*dict*) [source]

`move_relative_with_wait_option`(*dict*, *wait_until_done=False*) [source]

PI move relative method

Lots of implementation details in here, should be replaced by a facade

`read_position`() [source]

`send_position`() [source]

`set_filter`(*filterstring*) [source]

`set_zoom`(*zoomstring*) [source]

This should also signal back to the GUI that sth was changed

`sig_move_rel_xy`

`sig_zero_xy`

`stop_movement`() [source]

`unload_sample`() [source]

`zero_focus`() [source]

`zero_rot`() [source]

`zero_xy`() [source]

`zero_xyz`() [source]

`zero_z`() [source]

class `mesoSPIM.legacy.mesoSPIM.mesoSPIM_control.`**`mesoSPIM_Stages`**(*config*) [source]

Bases: **`object`**

Implements a facade providing a unified interface for the stages & movement depending on whether a Galil & L&N or Physik Instrumente stage combination is installed.

`move_absolute`() [source]

`move_relative`() [source]

`read_coordinates`() [source]

class `mesoSPIM.legacy.mesoSPIM.mesoSPIM_control.`**`mesoSPIM_State`**(*cfg*) [source]

Bases: **`PyQt5.QtCore.QObject`**

This class contains the microscope state

Here, we convert from a dictionary to a normal object

Any access to this global state should be locked via mutexes

TODO: Turn this into a singleton at some point, for now: Instantiate only once.

`calculate_samples`() [source]

class `mesoSPIM.legacy.mesoSPIM.mesoSPIM_control.`**`mesoSPIM_camera_GUI`** [source]

Bases: **`PyQt5.QtWidgets.QWidget`**

`display_status_message`(*string*, *time=0*) [source]

Displays a message in the status bar for a time in ms

If time=0, the message will stay.

`draw_crosshairs`() [source]

`set_image`(*image*) [source]