General notes on GladOS-Napari

Updates:

* 2024-05-07: initialisation

# (MM)-Controls tab

Controlled via **MMcontrols.py**.

For the tab, a dockWidget\_MMcontrol() is created, which starts a MMConfigUI() class. This has keyword arguments to show certain options (i.e. configugations, stages, ROIs, live-mode, etc).

Generally, these are all ‘semi-hardcoded’ to directly push changes via PycroManager to MicroManager.

Five main code-regions:

1. **Live mode**: has the info to start/stop live mode. In practice, just sets shared\_data.liveMode to True (or False), and just before that, sets core.set\_exposure() to the exposure time in ms
2. **ROI**: set/change the region-of-interest of the camera. Simply has options to reset the ROI (core.clear\_roi(), and zooming the ROI to the center half size (zoomROI, setROI)
3. **Stages**: all movement of XY, and one-dimensional (such as Z) stages. Generally, the layout is created and contains the XYstageLayout() and oneDstageLayout().
   1. **XYstageLayout** takes the current xy device (core.get\_xy\_stage\_device()), and adds arrows in a grid-like pattern to allow the user to change the stage – also has lineEdits to set the value how much each arrow the stage should move. In principle, this finally calls moveXYStage(), which simply calls core.set\_realtive\_xy\_position(deltaX,deltaY), to move the stage relatively to the current position.
   2. **oneDstageLayout** is similar to XY stage devices, except there are multiple QGridLayouts for each one-D-stage found (‘getDevicesOfDeviceType(‘StageDevice’)’), which contains arrows to move the stage and LineEdits to change the value of how much to move. I.e. there can be 5 QGridLayouts in a QStackedWidget, of which 4 are hidden, and they are loaded/hidden based on the dropdown. The stage is moved via moveOneDStage().
4. **MM-configs** shows all the ‘configuration settings’ (groups/presets) which are present in Micromanager. Each group is a row (addRow()), which has a label (addLabel()), and a dropdown, slider, or inputField (inputField should still be implemented). The current values can be updated via updateValueFromMM()/updateConfigsFromMM()
5. **General** has ‘general’ options, i.e. an ‘update all info from MM’, which also does the loading of data, and the ‘store all data’, which calls utils.CustomMainWindow.save\_state\_MMControls().

# Multi-D acquisition

Controlled via MDAGlados.py.

For the tab, a dockWidget\_MDA() is created, which starts a MDAGlados() class, with keywords arguments to instantiate the MDA (i.e. pre-set channels etc).

The MDAGlados has a goal to create a MDA object based on the settings in the GUI. It’s basically a giant wrapper for the pycromanager multi\_d\_acquisition\_events() function. The MDAGlados also pushes the core exposure and shared\_data.mdaModeParams when pressing the ‘acquire’ button.

Four main code-regions:

1. GUI: settings that the GUI uses to display itself. Most importantly, it allows for only showing part of the full MDA GUI, i.e. only showing the channels window.
2. Multi-D acquisition logic: The logci surrounding getting the MDA parameters and pushing these to shared\_data.
3. Properties: a somewhat dummy region to help with the GUI grid
4. List Widgets: Interactive List widgets for the XY and Channel Lists to allow for nice user-GUIs to add/move/remove positions in a channel or XY list.