ImageViewer

Release 1.0.0

Melissa Lajtos

Nov 22, 2020

CONTENTS:

CHAPTER

ONE

USAGE DESCRIPTION

1.1 Installation and start

To install the package, open a command or terminal window and cd into the root directory where `setup.py` is located, then enter:

```
python setup.py install
```

For this you need to have Python (3.6 or higher) installed. The required packages will be installed automatically, however, you might need to install `Qt5` (download at www.qt.io).

To start the programme, enter:

```
python run
```

1.2 How to use

1.2.1 GUI

Once you have started the programme, a window opens which should look like this:

Once a file was loaded, the image data of the file will be displayed and a few GUI elements will change, as can be seen in the following screenshot:

The elements highlighted here are:

1 - Main menu Click here to load a file or show metadata of a loaded file.

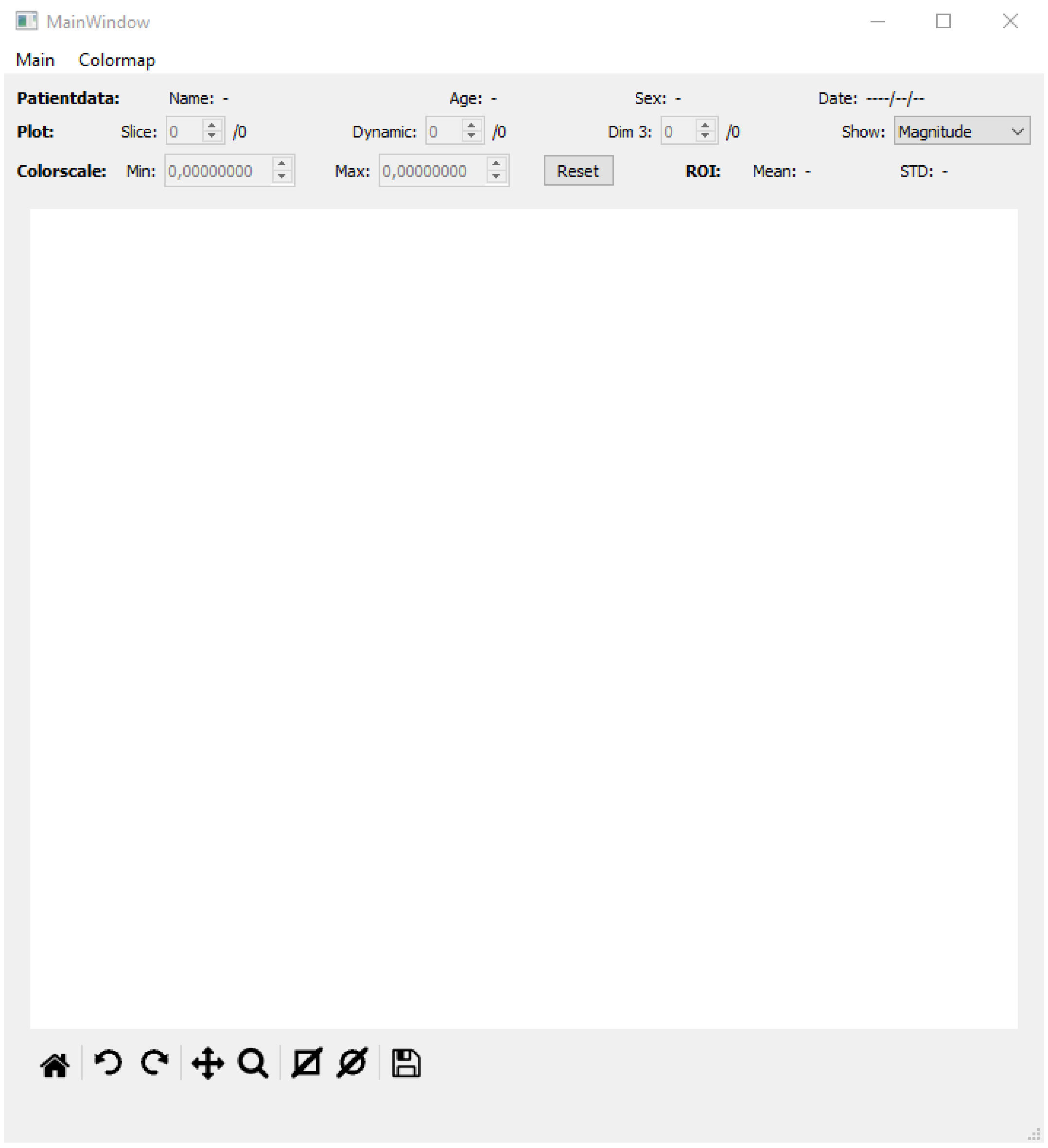
2 - Colormap menu Click here to change the colormap used for the plot.

3 - Patientdata information Some of the loaded file's metadata is displayed here, if given.

4 - Plot actions The slice, the dynamic, a 3rd dimension (if given) of the loaded dataset can be selected here for display. A dropbox allows changing between magnitude and phase of the data.

The slice and the dynamic can also be changed using the scroll wheel or the arrow keys: Up and down changes the slice, left and right changes the dynamic.

5 - Color scale settings Minimum and maximum of the color scale used for the plot can be changed here. When loading a file, the minimum and maximum of the color scale will be set to minimum and maximum of the loaded data (at the selected Dim 3). The reset button sets the limits back to those initial values.



Patientdata: Name: - Age: - Sex: - Date: ----/--/--

Plot: Slice: 0 /0 Dynamic: 0 /0 Dim 3: 0 /0 Show: Magnitude

Colorscale: Min: 0,00000000 Max: 0,00000000 ROI: Mean: - STD: -

Fig. 1: ImageViewer after opening.

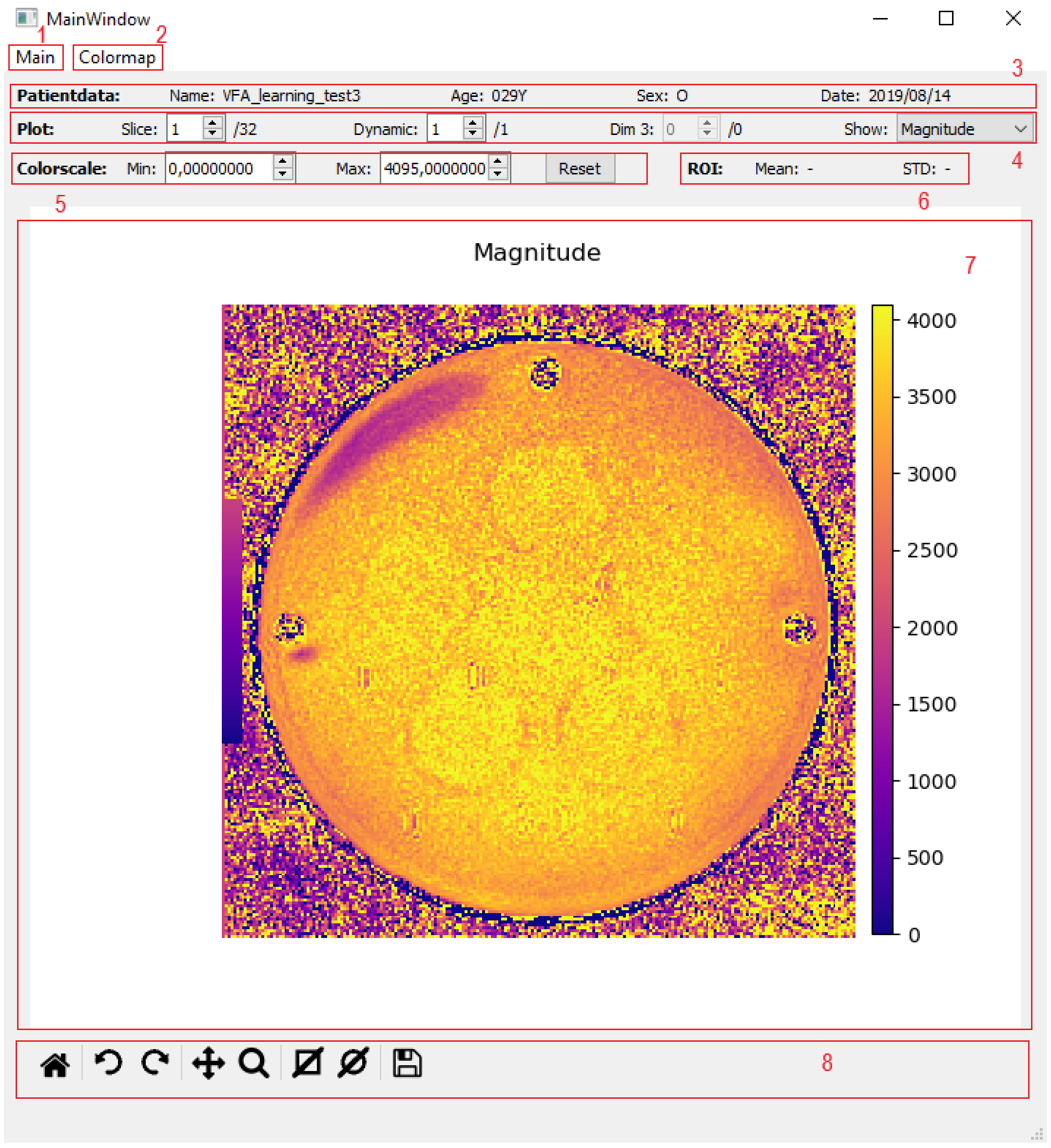


Fig. 2: GUI after file has been loaded.

The colorscale can also be changed by pressing the scroll wheel and moving the mouse: Vertical movement changes the distance between the limits, making the interval smaller or bigger. Horizontal movement moves the whole interval up or down, leaving the width of the interval unchanged.

6 - ROI values If a region of interest (ROI) was selected using the toolbar functions, the mean and standard deviation of the data currently displayed within that ROI are displayed here.

7 - Image canvas The plot of the selected data, as well as the corresponding colorbar are shown here. The following key shortcuts can be used to control the plot:

- The + and − keys allow zooming in and out.
- `ctrl` + any of the arrow keys allow moving through the plot (this only works if only a part of the plot is visible at the moment, e.g. after zooming in).

8 - Toolbar This augmented maplotlib toolbar holds the following functions, which are explained further in *Toolbar actions*:

- *Home*: Reset the plot.
- *Rotation (anti-)clockwise*: Clicking these two buttons rotates the plot by 90 degrees.
- *Pan/Zoom*: Pan (move) or zoom the plot using these built-in matplotlib actions.
- *ROI selection*: Select a region of interest in the shape of a rectangle or an ellipse inside the plot of which to calculate mean and standard deviation after pressing one of these buttons. Only one ROI can exist at a time.
- *Save figure*: Built-in matplotlib function to save the plot.

1.2.2 Data formats

The ImageViewer supports h5 and dicom files. Image as well as meta data can be read in both cases.

For **dicom** files it is important that certain naming rules are being followed in order to correctly identify and load different datasets and slices and dynamics of each dataset:

1. The first to -21 characters, which usually contain the scan name and the scan ID, are identical for each file belonging to the same set, and are unique among different sets.
2. Characters -16 to -13 contain the slice number.
3. The last 4 characters (-4 to -1) contain the dynamic number.

The negative numbers here correspond to counting “from the right” with -1 being the last character before the file extension *.dcm*. A working example is

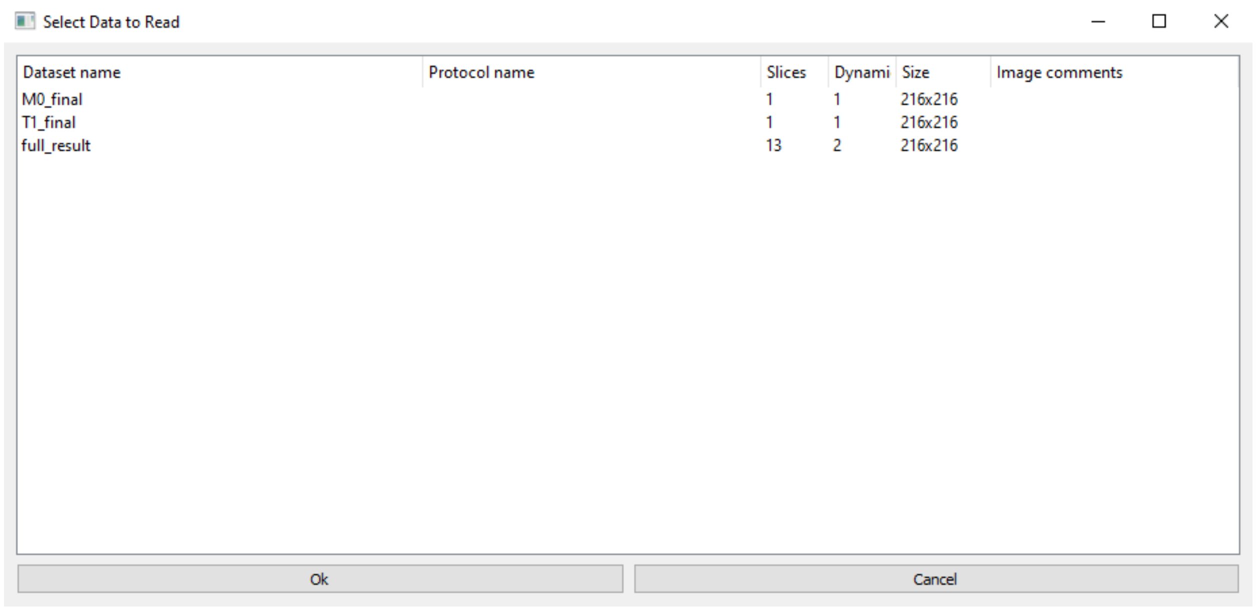
Datasetname_ex00090004000500010002.DCM

Datasetname_ex defines the dataset, *0004* indicates that this is the fourth slice, *0002* indicates that this is the second dynamic. All other characters and dimensions of the data are ignored (as of now), so 4 dimensions of image data are supported.

For **h5** files it is assumed that the last two dimensions of the image data refer to x and y coordinates. If the data has more than 2 dimensions, the first dimension is interpreted as the slice, the second as the dynamic. If there are 5 dimensions, the third dimension can also be selected in the GUI, but there is no specific name for it.

When selecting a dicom folder or .h5 file for loading which holds more than one dataset, another window will open, which lets the user select a dataset:

In that window, Size means the size of one image in pixels, Protocol name and Image comments refer to metadata fields.



Dataset name	Protocol name	Slices	Dynami	Size	Image comments
M0_final		1	1	216x216	
T1_final		1	1	216x216	
full_result		13	2	216x216	

Fig. 3: Window for selection of dataset.

1.2.3 Metadata

Some of the metadata is shown directly in the main window after a file was loaded.

In the case of dicom files, this refers to the fields *PatientName*, *PatientAge*, *PatientSex*, and *AcquisitionDate*. In case of h5 files, the `attrs` are searched for the keys

- *name, patient_name, patientname*
- *age, patient_age, patientage*
- *sex, patient_sex, patientsex*
- *date, acquisition_date, acquisitiondate*

where capitalisation is ignored, in order to retrieve that information.

For the dataset selection window, information is drawn from the fields *ProtocolName* and *ImageComments* in case of dicom, while in the .h5 file's `attrs` the keys being looked for are

- *protocol, protocol_name, protocolname*
- *comment, comments, image_comment, image_comments, imagecomment, imagecomments*

where, again, capitalisation does not matter.

In order to see all metadata available, an own metadata window can be shown by clicking *Metadata* in the main dropdown menu. The window has a search input field which lets you search for fields by name.

1.2.4 Toolbar actions

The available toolbar functions have already been mentioned briefly in section *GUI* and shall be explained further in the following paragraphs.

The **home** button resets the plot. Color scale limits, zooming and panning settings will return to initial values, any ROI will be deleted. Rotation as well as slice and dynamic selections remain.

The **rotation** buttons allow rotating the image either clock- or anti-clockwise.

The **pan** and **zoom** actions are default matplotlib functionalities. *Pan* allows dragging the image moving the mouse while pressing the left button, as well as zooming into or out of the point the cursor was at when the right mouse button was clicked while holding the button and moving the mouse up/right or down/left. *Zoom* allows zooming in to a rectangle drawn using the left mouse button, and out to a rectangle using the right mouse button.

For **ROI selection** there are two options: rectangle selection and ellipse selection. While keeping the left mouse button pressed, a region of interest can be drawn on the plot. The mean value and standard deviation of all datapoints within this ROI are calculated and displayed above the plot after the ROI was drawn. Once a ROI is created, it can be resized and moved around the plot.

Last is the **save figure** action, which again provides default matplotlib functionality. It allows for saving the current plot (including the colorbar).

CHAPTER

TWO

CODE DOCUMENTATION

This section covers the documentation of the source code. If you are just interested in using the ImageViewer, this part is not relevant for you.

2.1 Submodules

2.1.1 main module

class imageviewer.main.**DataHandler**

Bases: object

Class for storing image data.

active_data is the attribute the program will usually work with, and is always equal to either magn_data or phase_data. The data arrays stored in these variables are always 4-dimensional. The 4 dimensions are:

1. Slices
2. Dynamics
3. x
4. y

The original_data array is either 4- or 5-dimensional, see also *add_data()* for more information.

Variables

- **original_data** (*numpy.ndarray*) – Contains the original image data from the file (squeezed if there was an unnecessary dimension).
- **magn_data** (*numpy.ndarray*) – The magnitude values of the image data.
- **phase_data** (*numpy.ndarray*) – The phase values of the image data.
- **active_data** (*numpy.ndarray*) – Contains either magnitude or phase data, depending on magnitude.
- **active_min** (*float*) – Minimum value of active data.
- **active_max** (*float*) – Maximum value of active data.
- **magnitude** (*bool*) – Indicates whether magnitude or phase of data is currently selected by the user. Defaults to True.
- **empty** (*bool*) – Indicates whether data is currently loaded. Defaults to True.

add_data(*data*, *slices=1*, *dynamics=1*)
This function takes the image data from a loaded file, processes it, and stores it in the right attributes.

The number of dimensions gets checked:

- If 2, the data contains only one slice and one dynamic and will be expanded by two dimensions before being stored in order to handle it the same as 4-dimensional data.
- If 3, it is checked if the data contains multiple slices or multiple dynamics by looking at the parameters and will be expanded by one dimension before it is further processed.
- If 4, which is the desired number of dimensions, its magnitude and phase get stored in `magn_data` and `phase_data` directly.
- If there are 5 dimensions (even after squeezing), the first index of the third dimension is selected by default, so that `active_data` holds only 4 dimensions, while `original_data` holds all 5 dimensions.

It is important that the dimensions of `data` follow the order *slices, dynamics, x, y*.

Depending on the value of `magnitude`, either the magnitude or phase data gets stored in `active_data`.

Parameters

- **data** (*numpy.ndarray*) – Image data loaded from file.
- **slices** (*int*) – Number of slices `data` contains. Defaults to 1.
- **dynamics** (*int*) – Number of dynamics `data` contains. Defaults to 1.

change_active_data(*dim3=None*)
Responsible for setting `active_data`, which is used for plotting.

Changes the value of `active_data` to either `magn_data` or `phase_data` depending on the value of attribute `magnitude`. Also changes `active_min` and `active_max` accordingly.

If `dim3` is given, `magn_data` and `phase_data` are changed to absolute and phase values of `original_data` [:, :, dim3, :, :] respectively first.

clear_data()
Sets all data attributes back to None as they were after initialization.

rotate_data(*k*)
Rotates data (`magn_data`, `phase_data`, and `active_data`) at axes (-2, -1).

Parameters **k** (*int*) – Specifies how often the data is rotated by 90 degrees in anti-clockwise direction.

class imageviewer.main.**ImageViewer**
Bases: `PyQt5.QtWidgets.QMainWindow`, *imageviewer.ui.mainWindow.Ui_MainWindow*

Main class for showing the UI. Runs in the main thread.

Lets user open h5 and dicom files of which image data will be displayed and metadata can be shown.

Variables

- **filename** (`h5py._hl.files.File`, or list[str]) – Either the h5 file the user selected, or a list of the names of the first files of all dicom dataset within a dicom directory.
- **filetype** (*str*) – Indicates which filetype was loaded; either 'h5' or 'dicom'.
- **directory** (*str*) – Whole path of the dicom directory the user selected (including trailing slash '/').

- **dicom_sets** (*list[dict]*) – One dictionary (with keys *name*, *slices*, *dynamics*) for each dataset within the dicom directory identified by *IdentifyDatasetsDicom*.
- **dicom_ref** (*str*) – Filename (incl. path) of reference file of selected dicom set.
- **slice** (*int*) – The index of the slice of the image data being displayed.
- **dynamic** (*int*) – The index of the dynamic of the image data being displayed.
- **dim3** (*int*) – The index of the 3rd dimension of the image data being displayed.
- **mean** (*float*) – The mean value of the data inside the current selector (roi) of *NavigationToolbar*.
- **std** (*float*) – The standard deviation of the data inside the current selector (roi) of *NavigationToolbar*.
- **data_handler** (*DataHandler*) – Image data is being processed and stored here.
- **metadata_window** (*MetadataWindow*) – Window to show metadata of loaded file.
- **mplWidget** (*MplWidget*) – Widget used to visualize image data.
- **select_box** (*SelectBox*) – Window which lets user select a dataset within a selected file/directory.

add_data(*data*, *slices=1*, *dynamics=1*)

Hands the data over to *DataHandler* to store it appropriately by calling it's method *add_data()*.

Before that, *clear()* is called.

Parameters

- **data** (*numpy.ndarray*) – Image data from file.
- **slices** (*int*) – Number of slices `data` contains. Defaults to 1.
- **dynamics** (*int*) – Number of dynamics `data` contains. Defaults to 1.

after_data_added()

Takes care of enabling input fields and setting labels, before calling *MplWidget.create_plot()*.

Gets called after data was loaded from a file and added using *add_data()*.

browse_folder_dcm()

Opens a file dialog for selecting a dicom folder.

Once a folder is selected, it is stored in `directory` and `filetype` is set. If there is more than one file present within this directory (which is usually the case), a new thread, started by *IdentifyDatasetsDicom*, will identify the datasets. Once it is done, it will call *open_file_dcm()*.

If there is only one dicom file present in the directory (very untypically), this file is loaded directly using *GetFileContentDicom* which will run in a new thread to get the data within the file and call *add_data()* and *after_data_added()* when finished. Some attributes are also set directly, so other functions can be used either way.

browse_folder_h5()

Opens a file dialog for selecting an .h5 file.

Once a file is selected, it is stored in `filename`, `filetype` is set and *open_file_h5()* gets called.

change_cmap()

Calls *MplWidget.change_cmap()*.

change_cmax()
Calls *MplWidget.change_cmax()*.

change_cmin()
Calls *MplWidget.change_cmin()*.

change_dim3(*d*)
Changes the current index of 5th dimension of data (if not out of range for the current dataset).

Calls *DataHandler.change_active_data()* and *update_plot()*.

Parameters **d** (*int*) – The difference between new and old dim3 number.

change_dynamic(*d*)
Changes the current dynamic of data (if not out of range for the current dataset).

Calls *set_dynamic_spinbox()* and *update_plot()*.

Parameters **d** (*int*) – The difference between new and old dynamic number.

change_magn_phase()
Handles changing from magnitude to phase display and vice versa.

Is called when user changes the value of the comboBox in the GUI regarding magnitude and phase. Sets attribute `magnitude` to True when user selected *Magnitude*, sets it to False when user selected *Phase*. Calls *DataHandler.change_active_data()* and *update_plot()* afterwards. The colorscale limits and spin boxes get adjusted too.

change_slice(*d*)
Changes the current slice of data (if not out of range for the current dataset).

Calls *set_slice_spinbox()* and *update_plot()*.

Parameters **d** (*int*) – The difference between new and old slice number.

close()
Exits the application.

closeEvent(*event*)
Calls *close()*.

Parameters **event** (QCloseEvent:) – PyQt close event.

dim3_value_changed()
Gets called when value inside the dim3 spin box was changed. Calls *change_dim3()*.

dynamic_value_changed()
Gets called when value inside the dynamic spin box was changed. Calls *change_dynamic()*.

keyPressEvent(*event*)
Handles key press inputs.

Parameters **event** (QKeyEvent) – PyQt key input event.

mousePressEvent(*event*)
Sets focus on self.

Parameters **event** (QMouseEvent) – PyQt mouse input event.

open_file_dcm(*file_sets*)
Handles opening of dicom datasets after folder was selected.

Checks if there is more than one dataset within `file_sets`. If yes, opens instance of *SelectBox* which lets user select a dataset; if no, directly loads the data of the only dataset using *GetFileContentDicom*, which will call *add_data()* and *after_data_added()* when finished.

It sets attribute `dicom_sets` to `file_sets` and attribute `filename` to a list of dictionaries with the names of the first files, the #slices, and the #dynamics for each fileset.

Parameters **`file_sets`** (*`list[dict]`*) – A list that contains a dictionary which holds filename, #slices, #dynamics for each fileset.

`open_file_h5``()`

Handles opening/selecting of h5 dataset after file was selected.

Checks if there is more than one dataset within the file (attribute `filename`) to open. If yes, opens instance of *`SelectBox`* which lets user select a dataset and will call *`read_data()`*; if no, creates instance of *`GetFileContentH5`* which will run in a new thread to get the data within the file and call *`add_data()`* and *`after_data_added()`* when finished.

`read_data``()`

Handles the reading of a dicom or h5 dataset which was selected.

Depending on attribute `filetype`, the suiting thread (*`GetFileContentH5`* or *`GetFileContentDicom`*) is created to load the data. Methods *`add_data()`* and *`after_data_added()`* are called.

`reset_colorscale_limits``()`

Sets colorscale limits to actual minimum and maximum of currently selected dataset, `DataHandler.active_min` and `DataHandler.active_max`.

`reset_statistics``()`

Sets statistics (mean and std) values and GUI labels back to default.

`set_dynamic_spinbox``()`

Sets the spin box for current dynamic (`spinBox_dynamic`) to according value.

`set_patientdata_labels``()`

Sets the text values of the labels regarding patient data to metadata of read file, if metadata given.

`set_slice_spinbox``()`

Sets the spin box for current slice (`spinBox_slice`) to according value.

`show_metadata``()`

Calls *`MetadataWindow.open()`*.

`slice_value_changed``()`

Gets called when value inside the slice spin box was changed. Calls *`change_slice()`*.

`statistics``(`*startposition*, *endposition*, *selector*`)`

Calculates mean and std of data within a ROI.

Calculates mean and std of `DataHandler.active_data` within the patch defined by `selector` and `startposition` (upper left corner) and `endposition` (lower right corner). Changes the GUI labels' text values accordingly.

Parameters

- **`startposition`** (*`tuple[numpy.float64]`*) – Coordinates of top left corner.
- **`endposition`** (*`tuple[numpy.float64]`*) – Coordinates of bottom right corner.
- **`selector`** (*`str`*) – Type of selector (*rectangle* or *ellipse*).

`update_plot``()`

Calls *`MplWidget.update_plot()`*.

`wheelEvent``(`*event*`)`

Enables going through the data slices and dynamics using the mouse wheel.

A 120° turn in the y direction is turned into a slice difference of 1 and *change_slice()* is called. A 120° turn in the x direction is turned into a dynamic difference of -1 and *change_dynamic()* is called.

Parameters **event** (`QWheelEvent`) – The wheel event which contains parameters that describe a wheel event.

class `imageviewer.main.`**MetadataWindow**

Bases: `PyQt5.QtWidgets.QMainWindow`, *imageviewer.ui.metadataWindow.Ui_MainWindow*

Window for showing metadata of loaded files.

Variables

- **treeWidget** (*QTreeWidget*) – Widget which is used to list all metadata instances. Its 4 columns are Tag, Name, VR, Value.
- **lineEdit** (*QLineEdit*) – Input field used for searching metadata by name.

cancel()

Clears attribute `treeWidget` and closes the window.

filter()

Hides all items in attribute `treeWidget` whose names do not include the current text in attribute `lineEdit`.

open(*file*, *filetype*)

Populates attribute `treeWidget` with the metadata of the given file and opens the window.

Parameters

- **file** (str, or `h5py._hl.dataset.DataSet`) – Full filename including path of the dicom file, or h5 dataset.
- **filetype** (*str*) – Indicates type of file; either 'h5' or 'dicom'.

class `imageviewer.main.`**SelectBox**

Bases: `PyQt5.QtWidgets.QMainWindow`, *imageviewer.ui.selectBox.Ui_MainWindow*

Window for selecting the desired dataset within an h5 file or dicom folder.

Variables

- **treeWidget** (*QTreeWidget*) – Widget used to list all datasets to choose from. Has 4 columns: Dataset name, slices, dynamics, size.
- **selected** (*str*) – Name of the dataset selected in the UI window.

cancel()

Closes the window.

Clears attribute `treeWidget` and sets attribute `selected` to *None*.

confirm()

Stores the scan name and scan ID of the selected dataset in attribute `selected` and closes the window.

2.1.2 fileHandling module

class `imageviewer.fileHandling.`**`GetFileContent`**(*selected*)

Bases: `PyQt5.QtCore.QRunnable`

This class serves as a parent class for other classes which will handle loading data from different file types. It inherits from QRunnable, thus it will be called in a separate thread.

Parameters **`selected`** (*`str`*) – The name of the selected dataset data shall be loaded from.

class `imageviewer.fileHandling.`**`GetFileContentDicom`**(*file_sets*, *selected*, *directory*)

Bases: *`imageviewer.fileHandling.GetFileContent`*

Class for loading dicom image data. Inherits from *`GetFileContent`*.

Parameters

- **`file_sets`** (*`list[dict]`*) – Filesets identified by *`IdentifyDatasetsDicom`*.
- **`selected`** (*`str`*) – The scan name and ID of the selected dataset within the directory.
- **`directory`** (*`str`*) – The directory containing dicom files to read.

run()

Responsible for loading image data of a dicom dataset.

Loads the image data of a selected dataset into an array. The data array and the numbers of slices and dynamics are emitted with the `signals.add_data` signal. The signal `signal.finished` is emitted afterwards.

To load the slices and dynamics correctly, it is important that the filenames are named in a way that the slices number comes before the dynamics number.

Gets called when the thread is started.

class `imageviewer.fileHandling.`**`GetFileContentH5`**(*filename*, *selected*)

Bases: *`imageviewer.fileHandling.GetFileContent`*

Class for loading h5 image data. Inherits from *`GetFileContent`*.

Parameters

- **`filename`** (*`h5py._hl.files.File`*) – The selected .h5 file.
- **`selected`** (*`str`*) – The name of the selected dataset within the file. If the file only contains one dataset, this needs to be the same as parameter `filename`.

run()

Responsible for loading image data of .h5 file.

Loads the image data of a selected dataset into an array. Determines the number of slices and the number of dynamics by simply looking at the number of dimensions and the shape of the data under the following assumptions:

1. If the data has 3 dimensions, there are multiple slices, represented by the first dimension, and only one dynamic.
2. If the data has more than 3 dimensions, the first dimension represents slices, the second dimension represents dynamics.

The data array and the numbers of slices and dynamics are emitted with the `signals.add_data` signal. The signal `signals.finished` is emitted afterwards.

Gets called when the thread is started.

class `imageviewer.fileHandling.`**`GetFileContentSignals`**(**args: Any*, ***kwargs: Any*)

Bases: `PyQt5.QtCore.QObject`

Class for generating thread signals for *`GetFileContent`*.

`add_data`

alias of `builtins.object`

`finished`

Signal to emit when all is finished.

class `imageviewer.fileHandling.`**`IdentifyDatasetsDicom`**(*filenames*)

Bases: `PyQt5.QtCore.QRunnable`

Class for identifying files belonging together (forming a dataset) within a bunch of dicom files.

Inherits from QRunnable, thus it will be called in a separate thread.

Signals are from the *`IdentifyDatasetsDicomSignals`* class.

Parameters **`filenames`** (*`list[str]`*) – The names of all dicom files.

Variables **`filesets`** (*`list[dict]`*) – Dictionaries for all identified filesets, which hold filename, #slices, #dynamics.

`run`()

This function identifies the filesets formed by the files in parameter `filenames`.

The filename of the first file, the number of slices, and the number of dynamics are stored inside a dictionary for each fileset. All these dicts are stored in the list attribute `file_sets`, which is then passed on when the `signals.setsIdentified` signal is emitted.

The following naming conventions for files are important for this function to work (assuming the file ending '.dcm' is included):

1. The first to -25 characters, which usually contain the scan name and the scan ID, are identical for each file belonging to the same set, and are unique among different sets.
2. Characters -20 to -17 contain the slice number.
3. Characters -8 to -5 contain the dynamic number.

Gets called when the thread is started.

class `imageviewer.fileHandling.`**`IdentifyDatasetsDicomSignals`**(**args: Any*, ***kwargs: Any*)

Bases: `PyQt5.QtCore.QObject`

Class for generating thread signals for the *`IdentifyDatasetsDicom`* class.

`setsIdentified`

alias of `builtins.list`

2.2 Subpackages

2.2.1 imageviewer.ui package

The modules *mainWindow module*, *metadataWindow module*, and *selectBox module* have been created automatically from their .ui equivalents, which were created using the QTDesigner, and are therefore not documented properly. To convert a .ui to a .py file, simply run:

```
pyuic5 <filename>.ui -o <filename>.py
```

in the command line.

mplwidget module

class `imageviewer.ui.mplwidget.`**`MplWidget`**(*parent=None*)

Bases: `PyQt5.QtWidgets.QWidget`

Widget used to visualize image data.

A widget which holds a matplotlib canvas and a toolbar (*`NavigationToolbar`*) as attributes. Colormap and color limits can be changed, the plot can be zoomed and panned. Most of the actions however can be found in the toolbar.

Variables

- **`canvas`** (`matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg`) – The actual matplotlib figure canvas where data and colormap are plotted.
- **`toolbar`** (*`NavigationToolbar`*) – Toolbar with actions.
- **`empty`** (*`bool`*) – Indicates if canvas is empty.
- **`cmap`** (*`str`*) – Name of the colormap (matplotlib) used to plot the data. Defaults to ‘plasma’.
- **`im`** (`matplotlib.image.AxesImage`) – The image which gets displayed.
- **`color_min`** (*`float`*) – Minimum limit for color scale for the currently loaded data.
- **`color_max`** (*`float`*) – Maximum limit for color scale for the currently loaded data.
- **`imageViewer`** (*`ImageViewer`*) – Instance of the main window the widget is part of. Allows access to data and variables. It is set in *`ImageViewer`*’s __init__().

`canvasMouseMoveEvent`(*event*)

Used to overwrite the default `FigureCanvasQT.mouseMoveEvent()` method of attribute `canvas`.

Does what original method does and then calls own *`mouseMoveEvent()`* method.

Parameters **`event`** (`QMouseEvent`) – Instance of a PyQt input event.

`canvasMousePressEvent`(*event*)

Used to overwrite the default `FigureCanvasQT.mousePressEvent()` method of attribute `canvas`.

Does what original method does and then calls own *`mousePressEvent()`* method.

Parameters **`event`** (`QMouseEvent`) – Instance of a PyQt input event.

`change_cmap`(*cmap*)

Handles changing the colormap.

Sets attribute `cmap` to parameter `cmap`, changes colormap of the actual image and calls *`update_plot()`*.

Is called when user changes the colormap in the main window (*`ImageViewer`*).

Parameters **`cmap`** (*`str`*) – Name of new colormap.

change_cmax(*cmax*)
Handles changing the maximum color limit of the plot.

Changes attribute `color_max` to parameter `cmax` and updates the image shown, given that minimum would not be higher than maximum. In the other case attributes `cmin` and `cmax` would be set to the same value.

Parameters **cmax** (*float*) – New colormap maximum value.

change_cmin(*cmin*)
Handles changing the minimum color limit of the plot.

Changes attribute `color_min` to parameter `cmin` and updates the image shown, given that minimum would not be higher than maximum. In the other case attributes `cmin` and `cmax` would be set to the same value.

Parameters **cmin** (*float*) – New colormap minimum value.

clear()
Resets attributes to initial values and clears the canvas.

create_plot()
Used to create a plot on attribute `canvas` and set attributes for a dataset.

Clears `canvas.axes` and draws a new image on it. A matching colorbar is created on `canvas.axesc`. It is intended to use this method when a new dataset or file is loaded.

`canvas.axes.format_coord()` gets overwritten, so that data coordinates are shown in integer numbers. The selection mode (*rectselect* or *ellipseselect*) is also taken care of here (in case the button is pressed or there was a selector present used on the old image).

See also: *update_plot()*.

mouseMoveEvent(*event*)
Handles mouse moving while middle button (wheel) is being pressed.

Adjusts color range limits if movement direction is mainly vertical (upwards narrows the range, downwards widens it). Mainly horizontal movement moves the whole window of the color range (right movement sets it higher, left movement lower). *change_cmin()* and *change_cmax()* are triggered.

Parameters **event** (QMouseEvent) – Instance of a PyQt input event.

mousePressEvent(*event*)
Handles events caused by pressing mouse buttons.

Sets focus on main window (*ImageViewer*) if left mouse button was pressed.

Saves current cursor position when middle button (wheel) was pressed.

Parameters **event** (QMouseEvent) – Instance of a PyQt input event.

pan_plot(*direction*)
Allows panning plot in 4 main directions.

The distance (in pixels) by which the plot is panned depends on the current x and y limits of the plot, so that the plot is panned less after zooming in, and more after zooming out. Calls *update_plot()*.

Parameters **direction** (*str*) – Indicates direction to move plot to. Valid values are 'left', 'right', 'up', and 'down'.

update_plot()
Changes image data to currently active data and updates the plot and the colorbar.

The toolbar functions and settings remain as they are. It is intended to use this method when another image of the same dataset needs to be visualized (e.g. after colormap was changed or another slice was selected).

See also: `create_plot()`.

`zoom_plot`(*direction*)
Zooms in or out of the plot.

Calls `update_plot()`.

Parameters **`direction`** (`str`) – Indicates whether to zoom in or out. Valid values are 'in' and 'out'.

class `imageviewer.ui.mplwidget.`**`NavigationToolbar`**(**args*, ***kwargs*)
Bases: `matplotlib.backends.backend_qt5.NavigationToolbar2QT`

Custom matplotlib navigation toolbar used by `MplWidget`.

Enables matplotlib's default functionalities *home*, *pan*, *zoom*, *savefigure*, and adds new functionalities, which are selecting a region of interest (ROI), and rotating the plot by 90 degrees.

The class variable `toolitems` is overwritten so that some of matplotlibs default buttons and functionalities are removed. The method `_update_buttons_checked()` overwrites the parent method to include the self made *rectselect* and *ellipseselect* actions.

Variables **`toolitems`** (`tuple[tuple[str]]`) – List of toolitems to add to the toolbar, format of one toolitem is:

```
(
text, # the text of the button (often not visible to users)
tooltip_text, # the tooltip shown on hover (where possible)
image_file, # name of the image for the button (without the extension)
name_of_method, # name of the method in NavigationToolbar2 to call
)
```

`_update_buttons_checked`()
Syncs button checkstates to match active mode. Overwrites parent function to include *rectselect* and *ellipseselect* modes.

`activate_ellipse_select`()
Activates or deactivates *ellipseselect* mode. If needed, deactivates *pan* or *zoom*. Gets called when *ellipseselect* action is toggled.

`activate_rect_select`()
Activates or deactivates *rectselect* mode. If needed, deactivates *pan* or *zoom*. Gets called when *rectselect* action is toggled.

`create_ellipse_selector`()
Enables ellipse selection by creating an instance of `matplotlib.widgets.EllipseSelector`.

`create_icon`(*name*)
Creates a responsive icon with the style of default icons to be placed in the toolbar.

Parameters **`name`** (`str`) – Name (including relative path) of image file to be used.

Returns Icon for the toolbar.

Return type `PyQt5.QtGui.QIcon`

`create_rectangle_selector`()
Enables rectangular selection by creating an instance of `matplotlib.widgets.RectangleSelector`.

`deactivate_ellipse_selector`()
Deactivates *ellipseselect* mode and button/action. Gets called when *pan* or *zoom* action is toggled.

deactivate_hide_ellipse_selector()
: Calls `deactivate_ellipse_selector()` and hides the selector (in the GUI). Gets called when *rectselect* action is toggled.

deactivate_hide_rect_selector()
: Calls `deactivate_rect_selector()` and hides the selector (in the GUI). Gets called when *ellipseselect* action is toggled.

deactivate_rect_selector()
: Deactivates *rectselect* mode and button/action. Gets called when *pan* or *zoom* action is toggled.

home()
: Sets plot back to original view by calling `MplWidget.create_plot()`.

  All pan, zoom, selection, and colorscale limits settings are discarded, rotation not. Also calls `reset_statistics()` and `reset_colorscale_limits()`.

  Overwrites parent function.

on_ellipse_select(*eclick*, *erelease*)
: Gets called when the user completes an ellipse selection and emits a signal with the start and endpoints of the ellipse.

  Parameters

  - **eclick** (`matplotlib.backend_bases.MouseEvent`) – Matplotlib mouse click event, holds x and y coordinates.
  - **erelease** (`matplotlib.backend_bases.MouseEvent`) – Matplotlib mouse release event, holds x and y coordinates.

on_rect_select(*eclick*, *erelease*)
: Gets called when the user completes a rectangular selection and emits a signal with the start and endpoints of the rectangle.

  Parameters

  - **eclick** (`matplotlib.backend_bases.MouseEvent`) – Matplotlib mouse click event, holds x and y coordinates.
  - **erelease** (`matplotlib.backend_bases.MouseEvent`) – Matplotlib mouse release event, holds x and y coordinates.

rotate_anticlockwise()
: Rotates plot anti-clockwise once by calling `DataHandler.rotate_data()` and `MplWidget.update_plot()`.

rotate_clockwise()
: Rotates plot clockwise once by calling `DataHandler.rotate_data()` and `MplWidget.update_plot()`.

toolitems = (('Home', 'Reset original view', 'home', 'home'), (None, None, None, None)
: Overwritten parent attribute.

class imageviewer.ui.mplwidget.**NavigationToolbarSignals**(**args: Any*, ***kwargs: Any*)
: Bases: `PyQt5.QtCore.QObject`

  Class for generating thread signals for the `NavigationToolbar` class.

  roiSelection
  : alias of `builtins.tuple`

mainWindow module

class imageviewer.ui.mainWindow.**Ui_MainWindow**
Bases: object

retranslateUi(*MainWindow*)

setupUi(*MainWindow*)

metadataWindow module

class imageviewer.ui.metadataWindow.**Ui_MainWindow**
Bases: object

retranslateUi(*MainWindow*)

setupUi(*MainWindow*)

selectBox module

class imageviewer.ui.selectBox.**Ui_MainWindow**
Bases: object

retranslateUi(*MainWindow*)

setupUi(*MainWindow*)

2.2.2 imageviewer.tests package

The tests were created using the `unittest` framework. To run them open a command window and cd into the *imageviewer/tests* directory, then type:

```
python -m unittest test.py
```

to run all tests. If you want to run single tests instead of all, you can specify the command like this:

```
python -m unittest test.TestImageViewer
```

to run the tests of the `TestImageViewer` class.

test module

class imageviewer.tests.test.**TestDataHandling**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests *DataHandler*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Sets up the testing environment.

test_add_data_2dim()
Tests *imageviewer.main.DataHandler.add_data()* with 2-dimensional (one slice, one dynamic) data. Magnitude is wanted.

test_add_data_3dim()
Tests `add_data()` with 3-dimensional (multiple slices, one dynamic) data. Phase is wanted.

class imageviewer.tests.test.**TestFileLoad**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Tests file and data loading functionality of class `ImageViewer` and module `fileHandling`.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Sets up the testing environment.

test_identify_datasets_dicom()
Tests `run()`.

test_open_dicom()
Tests `open_file_dcm()` in the case of a single dicom fileset containing multiple files.

Since the method being tested also calls `add_data()` and `after_data_added()`, these methods are also being tested along the way. Normally, the function would start the thread of `GetFileContentDicom`, so the `run()` method of it is called manually and tested.

test_open_h5()
Tests `open_file_h5()` in the case of an .h5 file containing 3 sets, where the one selected has one slice.

Since the method being tested also calls `read_data()`, `add_data()`, and `after_data_added()`, these methods are also being tested along the way.

class imageviewer.tests.test.**TestImageViewer**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Class for testing basic settings and behaviour of the `ImageViewer` class.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Sets up the testing environment.

test_colormap_change()
Test setting different colormap by triggering action in menuColormap. Only one action should be checked at a time.

test_defaults()
Test default values that should be set when creating an instance of `ImageViewer`.

test_statistics_ellipse()
Tests `statistics()` in the case of an ellipse selector.

test_statistics_rectangle()
Tests `statistics()` in the case of a rectangle selector.

class imageviewer.tests.test.**TestMetadataWindow**(*methodName='runTest'*)
Bases: `unittest.case.TestCase`

Tests `MetadataWindow`.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Sets up the testing environment.

`test_open``()`

Tests *`open()`*.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX