# ImageViewer

***Release 1***

**Melissa Lajtos**

**Sep 14, 2020**

# CONTENTS:

CHAPTER

# ONE

# SUBMODULES

## 1.1 main module

**class** imageviewer.main.**DataHandling**
Bases: object

Image data is stored in this class sorted into magnitude and phase.

**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** (*True*) – Indicates whether data is currently loaded. Defaults to True.

**add_data**(*data*)
This function takes the data from a loaded file, processes it, and stores it in the right instance 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 assumed the data contains multiple slices but only one dynamic 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. Depending on the value of magnitude, the magnitude or phase data gets stored in active_data.

**Parameters** **data** (*numpy.ndarray*) – Image data loaded from file.

**change_active_data**()
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.

**clear_data**()
Sets all attributes back to 0 as they were after initialization.

**rotate_data**(*k*)
Rotates data (magn_data, phase_data, and active_data).

**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 using matplotlib, change the colormap, and select if magnitude or phase shall be displayed.

**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[list[str]]*) – List of lists with all files belonging to the same dataset within the dicom directory, identified by *IdentifyDatasetsDicom*.
- **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.
- **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*.
- **select_box** (*SelectBox*) – Window which lets user select a dataset within a selected file/directory.
- **data_handling** (*DataHandling*) – Data is being processed and stored here.
- **mplWidget** (*MplWidget*) – A self-made widget (inherits from QWidget) which is used to visualize image data.

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

Before that, *clear()* is called.

**Parameters** **data** (*numpy.ndarray*) – Image data from file.

**after_data_added**()
Takes care of enabling input fields and setting labels, before calling imageviewer.ui.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. If there is more than one file present within this directory (which is usually the case), a new thread, started by *IdentifyDatasetsDicom*, will sort the files into datasets. Once sorting 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* and also `set_patientdata_labels_dicom()` is called. Some attributes are also set directly, so the 'normal' file loading functions can be used.

**browse_folder_h5**()
Opens a file dialog for selecting an .h5 file.

Once a file is selected, it is stored in `filename` 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_dynamic**(*d*)
Changes the current dynamic of data (if not out of range for the current dataset).

**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 `magnitude` to True when user selected Magnitude, sets it to False when user selected Phase. Calls *DataHandling.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).

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

**close**()
Exits the application.

**closeEvent**(*event*)

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

**keyPressEvent**(*event*)
Handles key press inputs.

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

**mousePressEvent**(*event*)
Sets focus on self.

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

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

It sets `dicom_sets` to `file_sets` and `filename` to a list of the names of the first files of each fileset. These names will then stand for the set the file belongs to respectively.

**Parameters** **file_sets** (*list[list[str]]*) – A list that contains a list with the names of all files of a fileset for each fileset.

**open_file_h5**()
Handles opening of h5 datasets after file was selected.

Checks if there is more than one dataset within the file (filename) to open. If yes, opens instance of *SelectBox* which lets user select a dataset and will call *read_data()*; if no, calls set_patientdata_labels_h5() and creates instance of *GetFileContentH5* which will run in a new thread to get the data within the file.

**read_data**()
Handles the reading of a dicom or h5 dataset which was selected.

Checks if the file selected via *SelectBox* is of type h5 or dicom and starts the matching thread (GetFileContentH5 or GetFileContentDicom) to load the data and calls the right method (set_patientdata_labels_h5() or set_patientdata_labels_dicom()) to set the GUI labels' texts regarding patient data.

**reset_colorscale_limits**()
Sets colorscale limits to actual minimum and maximum of currently selected dataset.

**reset_statistics**()
Sets statistics (mean and std) values and 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 the data within the patch of the selector defined by startposition ( upper left corner) and endposition (lower right corner) and changes the GUI labels' text values accordingly.

**Parameters**

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

**update_plot**()
Calls *MplWidget.update_plot()*.

**wheelEvent**(*event*)
This function 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 :meth`change_slice` is called. A 120° turn in the x direction is turned into a dynamic difference of -1 and :meth`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 dicom files.

**Variables**

- **treeWidget** – Widget which is used to list all metadata instances. Its 4 columns are Tag, Name, VR, Value.
- **lineEdit** – Input field used for search terms.

**cancel**()
Closes the window.

**filter**()
Hides all items in treeWidget whose names do not include the current text in lineEdit.

**open**(*file*, *filetype*)
Populates treeWidget with the metadata of the given file and shows the window.

**Parameters**

- **file** (*str*) – Full filename including path of the dicom file to read metadata from.
- **filetype** – 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** – Widget which is used to list all datasets to choose from. Has 4 columns: Dataset name, slices, dynamics, size.
- **selected** (*None or str*) – Name of the selected file within the UI window.

**cancel**()
Closes the window and sets selected back to *None*.

**confirm**()
Stores the name of the selected dataset in selected and closes the window.

## 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 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 data. Inherits from *GetFileContent*.

**Parameters**

- **file_sets** (*list[list[str]]*) – Filesets identified by *IdentifyDatasetsDicom*.
- **selected** (*str*) – The name of the first file of the selected dataset within the directory.
- **directory** (*str*) – The selected directory.

**run**()
Loads the selected dataset into an array and passes it to self.signals.add_data.emit(). Emits finished signal after that.

Gets called when the thread is started.

**class** imageviewer.fileHandling.**GetFileContentH5**(*filename*, *selected*)
Bases: *imageviewer.fileHandling.GetFileContent*

Class for loading h5 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**()
Loads the selected dataset into an array and passes it to self.signals.add_data.emit(). Emits finished signal after that.

Gets called when the thread is started.

**class** imageviewer.fileHandling.**GetFileContentSignals**
Bases: PyQt5.QtCore.QObject

Class for generating thread signals for *GetFileContent*.

**add_data**

**finished**

**class** imageviewer.fileHandling.**IdentifyDatasetsDicom**(*filenames*)
Bases: PyQt5.QtCore.QRunnable

Class for identifying files belonging together (forming a dataset) within a dicom folder. It 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 the files within the dicom directory.

**run**()
This function looks at all filenames in parameter `filenames` separately and sorts them into sets.

It does so by comparing a filename to the next filename: if the filenames differ at at least two characters and the first and last different characters are more than 2 indices apart, they are considered to be of different sets. This works on the assumption that the filenames contain ongoing numbers, one representing dataset and one representing file (slice) within dataset. Using 2 indices as the criteria, this approach may be a problem when there are more than 1000 files within a set.

Gets called when the thread is started.

**class** imageviewer.fileHandling.**IdentifyDatasetsDicomSignals**
Bases: PyQt5.QtCore.QObject

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

**setsIdentified**

CHAPTER

TWO

# SUBPACKAGES

## 2.1 imageviewer.ui package

### 2.1.1 mainWindow module

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

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### 2.1.2 selectBox module

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

**retranslateUi**(*MainWindow*)

**setupUi**(*MainWindow*)

### 2.1.3 mplwidget module

**class** imageviewer.ui.mplwidget.**MplWidget**(*parent=None*)
Bases: PyQt5.QtWidgets.QWidget

Self-made widget used to visualize image data.

**Variables**

- **canvas** (matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg) – The actual matplotlib figure canvas where data and colormap are plotted.
- **toolbar** (*NavigationToolbar*) – Toolbar which holds 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. Will be used for clim parameter of im.

- **color_max** (*float*) – Maximum limit for color scale for the currently loaded data. Will be used for `clim` parameter of `im`.
- **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 `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 `canvas`.

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

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

**change_cmap**(*cmap*)
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*)
Changes `color_max` according to `cmax` and updates the image shown.

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

**change_cmin**(*cmin*)
Changes `color_min` according to `cmin` and updates the image shown.

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

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

**create_plot**()
Used to create a plot and set attributes for a dataset.

Clears `canvas.axes`, creates (new) image to show and draws it on `canvas`. 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 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 vertical (upwards narrows the range, downwards widens it). 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 because of this.

**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*)
Pans plot in 4 main directions.

**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.

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.

**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*.

The class variable *toolitems* is overwritten so that some of matplotlibs default buttons and functionalities are removed. The method *_update_buttons_checked()* is also overridden 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
)
```

**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**()
This function simply enables ellipse selection by creating an instance of matplotlib.widgets.EllipseSelector.

**create_rectangle_selector**()
This function simply 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. Gets called when *rectselect* action is toggled.

**deactivate_hide_rect_selector**()
Calls *deactivate_rect_selector()* and hides the selector. 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 *DataHandling.rotate_data()* and *MplWidget.update_plot()*.

**rotate_clockwise**()
Rotates plot clockwise once by calling *DataHandling.rotate_data()* and *MplWidget.update_plot()*.

**toolitems = (('Home', 'Reset original view', 'home', 'home'), (None, None, None, None)**

**class** imageviewer.ui.mplwidget.**NavigationToolbarSignals**
Bases: PyQt5.QtCore.QObject

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

**ellipseSelection**

**rectangularSelection**

## 2.2 imageviewer.tests package

### 2.2.1 test module

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

Tests *DataHandling*.

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**()

**test_add_data_2dim**()
Tests *add_data()* with one-sliced (2-dimensional) data. Magnitude is wanted.

**test_add_data_3dim**()
Tests *add_data()* with multi-sliced (3-dimensional) data. Phase is wanted.

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

Tests file and data loading functionality of *ImageViewer* and *imageviewer.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**()

**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 also called and tested.

**test_open_h5**()
Tests *open_file_h5()* in the case of a .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**()

**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

To test *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**()

**test_open**()
Tests *open()*.

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## i