ImageDisplayQT

Release 1.0.0

Mike Hughes

Mar 07, 2023

CONTENTS

ImageDisplayQT is a PyQT widget for live scientific image display in Python QT Graphical User Interfaces.

It was designed as a simple and lightweight widget that allows real-time display of images. Features include a status bar, with pixel value inspection, a draggable region of interest, zoom and pan, and several types of overlay. Monochrome images can be displayed using any colormap from matplotlib.

Install using:

```
pip install ImageDisplayQT
```

It is developed mainly by Mike Hughes's lab in the Applied Optics Group, School of Physics and Astronomy, University of Kent. The package was originally developed for GUIs for in endoscopic microscopy, including fluorescence endomicroscopy and holographic endomicroscopy.

The project is hosted on github. Bug reports, contributions and pull requests are welcome.

CHAPTER

ONE

CONTENTS

1.1 Getting Started

Import the `ImageDisplay` class:

```
from image_display import ImageDisplay
```

Create an instance of `ImageDisplay`:

```
imDisplay = ImageDisplay()
```

and then add this widget to your GUI, eg. using `addWidget`.

To update the image display, call:

```
imDisplay.set_image(img)
```

where `img` is either a 2D numpy array containing a monochrome image or a 3D numpy array containing a colour image with the third dimensions containing the red, green and blue channels.

Toggle the status bar visibility using `set_status_bar`, for example:

```
imDisplay.set_status_bar(True)
```

Toggle the zoom facility using the mouse scroll wheel (or pinch zoom) using `set_zoom_enabled`, for example:

```
imDisplay.set_zoom_enabled(True)
```

Once zoomed you can pan by holding the middle or right mouse buttons.

Toggle the ability to draw a rectangular region of interest by holding the left mouse button and dragging using `set_roi_enabled`, for example:

```
imDisplay.set_roi_enabled(True)
```

You can set the colormap to be any Matplotlib colormap, for example:

```
imDisplay.set_colormap('hsv')
```

You can choose whether or not the image intensity is autoscaled to use the full dynamic range using `autoscale_enabled`, for example:

```
imDisplay.autoscale_enabled(True)
```

Images are always displayed as 8bit images. If autoscale is set to `True` then the smallest and largest image pixel values will be mapped to 0 and 255 respectively. For colour images, all three channels are scaled in the same way.

If autoscale is set to `False` then no mapping will occur by default, the input image will simply be cast to an 8 bit unsigned image. This can be changed using `set_display_range`:

```
imDisplay.set_display_range(min, max)
```

Where image pixel values of `min` and below will be mapped to 0, and `max` and above to 255.

1.2 Example

An example is included in the `examples` folder of the github repository.

This example is a bare-bones PyQT GUI which displays images in sequence from either a monochrome or colour tif stack using the ImageDisplay widget. Various features can be toggled using a set of checkboxes.

1.3 Overlays

Five types of overlay are supported:

- Rectangle
- Ellipse
- Line
- Point
- Text

Overlays are added using the `add_overlay` function. This returns a reference to the overlay. This can be later passed to `remove_overlay` to delete the overlay. Alternatively, all overlays can be removed using `clear_overlays`.

To create a rectangular overlay:

```
from image_display import ImageDisplay
imDisplay = ImageDisplay()

imDisplay.add_overlay(ImageDisplay.RECTANGLE, x, y, w, h, pen, fill)
```

where `x` and `y` are the x and y co-ordinates of the top left of the rectangle, respectively, and `w` and `h` are the width and height. These co-ordinates are specified in terms of pixels in the original image (i.e. not screen pixels). Co-ordinates outside the image can be specified in which case the overlay will be clipped.

The `pen` and `fill` should be a `QPen` and `QBrush` respectively. For example:

```
overlay = imDisplay.add_overlay(ImageDisplay.RECTANGLE, 90, 100, 30, 40, QPen(Qt.blue, 2,
↪ Qt.SolidLine), QBrush(Qt.red))
```

generates a rectangle with a blue solid outer line of thickness 2 and a red fill. For a transparent fill, pass `None` for `fill`.

To add an ellipse, the structure is:

```
overlay = imDisplay.add_overlay(ImageDisplay.ELLIPSE, x, y, w, h, pen, fill)
```

For a line:

```
overlay = imDisplay.add_overlay(ImageDisplay.LINE, x, y, w, h, pen)
```

For a point:

```
overlay = imDisplay.add_overlay(ImageDisplay.POINT, x, y, pen)
```

Note that no `fill` is specified for lines and points.

To add a text overlay:

```
overlay = imDisplay.add_overlay(ImageDisplay.TEXT, x, y, pen, text)
```

where `text` is the string to add. The co-ordinates are the lower left of the bounding rectangle of the text box.

To remove any overlay, use:

```
imDisplay.remove_overlay(overlay)
```

where `overlay` is the reference returned when creating the overlay using `add_overlay`.

To clear all overlays use:

```
imDisplay.clear_overlays()
```

1.4 Customisation

The widget appearance and behaviour can be customised as follows without sub-classing.

1.4.1 Overall Appearance

At creation, the following stylesheet is applied to the widget:

```
setStyleSheet("border:1px solid white")
```

Apply a different stylesheet to change the overall appearance, e.g. to change or remove the border.

1.4.2 Status Bar

Set the visibility of the status bar (visible by default):

```
set_status_bar(True/False)
```

To customise colours of the status bar, the following values can be set directly:

- `statusPen` The border colour/style, provide a `QPen`, such as `QPen(Qt.white, 2, Qt.SolidLine)`.
- `statusBrush` The fill colour/style, provide a `QBrush`, such as `QBrush(Qt.white, Qt.SolidPattern)`.
- `statusTextPen` The font colour, provide a `QPen`, such as `QPen(Qt.black, 2, Qt.SolidLine)`.

1.4.3 Zoom

To enable or disable zooming (enabled by default):

```
set_zoom_enabled(True/False)
```

To control how much a mouse wheel scroll or pinch zoom changes the zoom, use:

```
set_zoom_step_divider(zoomDivider)
```

where a bigger value of `zoomDivider` makes the amount zoomed in by each mouse wheel click smaller. The zoom is (base 2) logarithmic. A value of 1 means the zoom will go as 1X, 2X, 4X, 8X, 16X etc. Default is 2.

Set the visibility of the zoom indicator using:

```
set_zoom_indicator_enabled(True/False)
```

To customise colours of the zoom indicator, the following values can be set directly:

- `zoomIndicatorPen` The colour of the zoom indicator, provide a `QPen` such as `QPen(Qt.white, 1, Qt.SolidLine)`.
- `zoomIndicatorBrush` The fill colour of the zoom indicator, provide a `QBrush` such as `QBrush(Qt.white, Qt.SolidPattern)`.
- `zoomIndicatorWidth` With of zoom indicator, default 60.
- `zoomIndicatorOffsetX` Horizontal position of zoom indicator, relative to top right corner of image, default 20.
- `zoomIndicatorOffsetY` Vertical position of zoom indicator, relative to top right corner of image, default 20.

1.4.4 Region of Interest (ROI)

Set whether a ROI can be dragged using:

```
set_roi_enabled(True/False)
```

To customise colours, the following values can be set directly:

- `roiDragContrastPen` The colour/style of the first rectangle to be drawn while the ROI is being dragged, provide a `QPen` such as `QPen(Qt.white, 2, Qt.SolidLine)`.
- `roiDragPen` The colour/style of the second rectangle to be drawn while the ROI is being dragged. This is drawn over the first rectangle and so should usually be a dotted/dashed line of a different colour to help improve visibility when colour images are displayed. Provide a `QPen` such as `QPen(Qt.red, 2, Qt.DotLine)`.
- `roiContrastPen` The colour/style of the first rectangle to be drawn of an ROI which has been set. Provide a `QPen` such as `Pen(Qt.white, 2, Qt.SolidLine)`.
- `roiPen` The colour/style of the second rectangle to be drawn of an ROI which has been set. This is drawn over the first rectangle and so should usually be a dotted/dashed line of a different colour to help improve visibility when colour images are displayed. Provide a `QPen` such as `QPen(Qt.green, 2, Qt.DotLine)`.
- genindex

Acknowledgements: Funding to Mike Hughes's lab from EPSRC (Ultrathin fluorescence microscope in a needle, EP/R019274/1), Royal Society (Ultrathin Inline Holographic Microscopy).