samuroi Documentation

Release 0.1

Martin Rueckl, Stephen Lenzi, Friedrich Johenning

September 19, 2016

Contents

Contents:

CHAPTER 1

Examples

This section presents some example code snippets which are ment to be run from within a Jupyter notebook. All examples can be found in the git repository in `doc/examples`. If you want to run SamuROI from within scripts, see *SamuROI from within normal python scripts*.

Lets get started with how to show the GUI window. Only a few lines are required to load a tif file and show up the gui:

```
# samuroi imports
from samuroi import SamuROIWindow
from samuroi.plugins.tif import load_tif

# load the data into a 3D numpy array
data = load_tif('/path/to/your/file.tif')

# show the gui for the loaded data
mainwindow = SamuROIWindow(data=data)

# maybe necessary depending on IPython
mainwindow.show()
```

1.1 The underlying SamuROIData object

SamuROI tries to follow a strict separation of data and GUI which is sometimes referred as Document/View or Model/View pattern (for more infos see the technical description in the paper and the documentation of `samuroi.SamuROIData`). When a SamuROI window is created, it also creates an underlying SamuROIData object which can be obtained by:

```
# get handle on the document of the main window
doc = mainwindow.segmentation
```

In the following examples we will show how one can read and manipulate many of the attributes of the `samuroi.SamuROIData` class.

1.1.1 Working with ROIs

All ROIs of a SamuROIData object are stored within the attribute masks of type `samuroi.maskset.MaskSet`. This class keeps track of the types of masks it stores and provides events for added and removed masks. It is iterable and checks that no mask is added twice (one can add multiple copies of a mask, but not the same object). Iteration over all masks is simple as

```
# iterate over all masks
for m in doc.masks:
   print m
```

Because the `samuroi.maskset.MaskSet` also introduces a hierarchy for the types of masks it stores, one can also iterate through all masks of a certain type like this:

```
# iterate over all branch masks
from samuroi.masks.branchmask import BranchMask
for bm in doc.masks[BranchMask]:
   print bm
```

1.1.2 Adding and removing masks

We can simply add and remove rois from the `samuroi.maskset.MaskSet`. E.g. we can add all masks from a swc file like this. (If you need this functionality you can also call `samuroi.SamuROIData.load_swc()`)

```
from samuroi.masks.circle import CircleMask
from samuroi.masks.branch import BranchMask
from samuroi.plugins.swc import load_swc
swc = load_swc("path/to/file.swc")
for b in swc.branches:
    if len(b) > 1:
        mask = BranchMask(data=b)
    else:
        mask = CircleMask(center=b[['x', 'y']][0], radius=b['radius'][0])
    # this will add the mask to the maskset and trigger maskset's added event.
    doc.masks.add(mask)
```

Removing masks is similar

```
from samuroi.masks.branch import BranchMask
# use try catch since there might not be any BranchMask in the MaskSet
try:
    # get handle on the "first" branch in the maskset
    first_branch = doc.masks[BranchMask].next()
    # remove the mask, will trigger the preremove and removed event of the masklist.
    doc.masks.discard(first_branch)
except:
    # we cant remove anything if its not there
    pass
```

Warning: Removing masks from the maskset from within an iteration through its elements may lead to undefined behaviour.

1.2 Installing a custom postprocessor

Often one wants to apply some custom postprocessor to traces produces by the ROIs. This can be achieved by installing a custom postprocessor. E.g. if you click on the “detrend” and “smoothen” buttons in the gui, respective postprocessors will transform the trace bevore it gets displayed in any widget. In this example we will show how we can transform the trace such that it has a zero mean over time.

```
def zero_mean_postprocessor(trace):
   """
```

```
  :param trace: a 1D numpy array holding the trace of the ROI.
  :return: a 1D numpy array with transformed trace.
  """
  return trace - numpy.mean(trace)

# change the postprocessor
doc.postprocessor = zero_mean_postprocessor
```

Pretty easy, huh? Let's try something more advanced:

todo build event detection postprocessor.

1.3 SamuROI from within normal python scripts

Ususally the IPython notebook takes care of some Qt mechanics that are required by SamuROI. Specifically this is: The Qt main event loop, which handles all direct user input on the GUI. Hence when one wants to run SamuROI from within a script, this handling has to be done by one self. The following example shows what is necessary and provides a nice starting point for your own applications :-)

```
import numpy

from samuroi.gui.samuroiwindow import SamuROIWindow
from samuroi.plugins.tif import load_tif
from samuroi.plugins.swc import load_swc
from samuroi.masks.segmentation import Segmentation as SegmentationMask

import sys
from PyQt4 import QtGui

import argparse

parser = argparse.ArgumentParser(description='Open SamuROI and load some data.')

parser.add_argument('filename', type=str, help='The filename of the tif file to use as data.')

parser.add_argument('--swc', dest='swcfiles', type=str, action='append', help='Filename of swc file t

parser.add_argument('--segmentation', dest='segmentations', type=str, action='append',
                    help='Filename of segmentations to load. (.npy files)')

if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    args = parser.parse_args()

    data = load_tif(args.filename)

    # show the gui for the filtered data
    mainwindow = SamuROIWindow(data=data)

    for filename in args.swcfiles:
        swc = load_swc(filename)
        mainwindow.segmentation.load_swc(swc)

    if args.segmentations is not None:
        for filename in args.segmentations:
            segdata = numpy.load(filename)
```

```
            seg = SegmentationMask(data=segdata, name="filename")
            mainwindow.segmentation.masks.add(seg)

    mainwindow.show()
    sys.exit(app.exec_())
```

CHAPTER 2

How ROIs work in SamuROI

Note: In the following we will use mask as synonym for ROI.

SamuROI comes with a set of predefined types of ROIs. Those are:

- **pixel based ROI (`samuroi.masks.pixel.PixelMask`):** This roi holds a list of pixels for which the trace is calculated as average over all those pixels.
- **circle shaped ROI (`samuroi.masks.circle.CircleMask`):** This roi is defined by a center and radius. The resulting trace takes all pixels into account that intersect with the circle.
- **branch ROI (`samuroi.masks.branch.BranchMask`):** This roi is intended for one dimensional structures within the video data. It provides functionality to be split into child sub segments. Child masks can be accessed with `samuroi.masks.branch.BranchMask.children`.
- **polygon ROI (`samuroi.masks.polygon.PolygonMask`):** As the name states, this class represents arbitrary polygon shapes.
- **segmentations (`samuroi.masks.segmentation.Segmentation`):** This class is rather special, since it is not a real ROI, but a set of rois defined by a 2D array with the same shape as the video resolution.

All of these classes inherit from a common base class `samuroi.masks.mask.Mask` which requires the derived classes to implement an "apply" function `samuroi.masks.mask.Mask.__call__()`. Whenever a trace of a roi is calculated this function will be called on the respective roi object.

The `samuroi.masks.branch.BranchMask` and the `samuroi.masks.segmentation.Segmentation` also have a set of child masks.

For examples how to work with these classes see *Working with ROIs*.

CHAPTER 3

samuroi package

3.1 Subpackages

3.1.1 samuroi.gui package

3.1.2 samuroi.masks package

3.1.3 samuroi.plugins package

3.1.4 samuroi.util package

3.2 The SamuROIWindow class

3.3 The SamuROIData class

3.3.1 The MaskSet class

CHAPTER 4

Indices and tables

- genindex
- modindex
- search