Contents

1 Overview

2 Documentation
   2.1 Installation .......................................................... 5
   2.2 Tutorial .............................................................. 7
   2.3 SIMA API .................................................................. 15
   2.4 ROI Buddy GUI ......................................................... 17
   2.5 Publications .......................................................... 22
   2.6 Credits .................................................................. 23
   2.7 How you can help .................................................... 24

3 Citing SIMA .................................................................. 27

4 Indices and tables .......................................................... 29
Release  1.3.0
Date  August 26, 2015
Email  software@losonczylab.org
Forum  https://groups.google.com/forum/#!forum/sima-users
GitHub  https://github.com/losonczylab/sima
SIMA (Sequential IMage Analysis) is an Open Source package for analysis of time-series imaging data arising from fluorescence microscopy. The functionality of this package includes:

- correction of motion artifacts
- segmentation of imaging fields into regions of interest (ROIs)
- extraction of dynamic signals from ROIs

The included ROI Buddy software provides a graphical user interface (GUI) supporting the following functionality:

- manual creation of ROIs
- editing of ROIs resulting from automated segmentation
- registration of ROIs across separate imaging sessions
2.1 Installation

We recommend that you perform a binary installation. You should not attempt to build SIMA if you are not familiar with compiling software from sources.

On Windows, we recommend using a scientific python distribution for installing the various prerequisites. Recommended distributions are:

- Python(x,y)
- WinPython
- Anaconda

For Mac OS X, we recommend installing the prerequisites, especially OpenCV, using a package manager, such as MacPorts.

2.1.2 Prerequisites

SIMA depends on various freely available, open source software. Whenever possible, we recommend installing these dependencies with your operating system’s or Python distribution’s package manager prior to installing SIMA.

- Python 2.7
- numpy >= 1.8
- scipy >= 0.13.0
• scikit-image >= 0.9.3
• scikit-learn >= 0.11
• shapely >= 1.2.14 (Windows users: be sure to install from Christophe Gohlke’s built wheels)
• pillow >= 2.6.1
• future >= 0.14

Depending on the features and data formats you wish to use, you may also need to install the following packages:

• OpenCV >= 2.4.8, required for segmentation, registration of ROIs across multiple datasets, and the ROI Buddy GUI
• picos >= 1.0.2, required for spike inference (>= 1.1 required for Python 3)
• pyfftw, allows faster performance of some motion correction methods when installed together with FFTW.
• h5py >= 2.2.1 (2.3.1 recommended), required for HDF5 file format
• bottleneck >= 0.8, for faster calculations
• matplotlib >= 1.2.1, for saving extraction summary plots
• mdp, required for ICA demixing of channels

If you build the package from source, you may also need:

• Cython

If you want to generate the documentation, you will also need:

• Sphinx >= 1.3.1

If you are using the spike inference feature, we strongly recommend installing MOSEK (free for academic use) which greatly speeds up the inference.

### 2.1.3 SIMA installation

#### Linux

The SIMA package can be installed from the python package index:

```
$ pip install sima --user
```

Source code can be downloaded from https://pypi.python.org/pypi/sima. If you download the source, you can install the package with setuptools:

```
$ python setup.py build
$ python setup.py install --user
```

#### Windows

The SIMA package can be installed from the python package index:

```
$ pip install sima
```

Alternatively, the packaged wheel is available at <https://pypi.python.org/pypi/sima> to be installed with your Python distribution’s package manager.

NOTE: You may need to install shapely separately from the package at: <http://www.lfd.uci.edu/~gohlke/pythonlibs/#shapely>
If building SIMA from source on Windows, you may also need to follow these instructions for compiling the Cython extensions.

**Mac OS X**

For installing the dependencies, we recommend using MacPorts. If you do not already have XCode installed, downloading XCode from the App Store, and then run the following commands in the Terminal to complete the XCode installation and license agreement:

```bash
$ xcode-select --install
$ gcc --version
```

Next, download and install MacPorts. Then run the following command in terminal to install SIMA and its dependencies:

```bash
$ sudo port selfupdate
$ sudo port select --set python python27
$ sudo port select --set pip pip27
$ pip install sima --user
```

### 2.2 Tutorial

Here we provide some basic usage examples for the SIMA Python package. These examples can be run in a standard Python shell, or with IPython. Users new to Python may wish to consult the Python documentation.

For more details on the classes and methods that comprise the SIMA package, please consult the SIMA API.
2.2.1 Importing SIMA

Like all Python packages, the SIMA package must be imported prior to use. Here we show a simple example of importing the SIMA package, which results in printing the docstring containing view basic information about the package.

```python
>>> import sima
```

In all future examples, we assume that the SIMA package has been imported as shown above.

Submodules of the SIMA package also need to be imported before use. For example, the motion correction module can be imported as follows.

```python
>>> import sima.motion
```

Individual classes or functions can be imported from submodules. For example, we can import the iterable object for use with multi-page TIFF files with the following command:

```python
>>> from sima.motion import HiddenMarkov2D
```

For more details on importing, consult the Python documentation.

2.2.2 Example data

The SIMA package comes with a small amount of example data that, although insufficient to allow for reasonable results from the motion correction and segmentation algorithms, can at least be used to run the functions. For the purposes of this tutorial, we copy the example data into the current working directory.

```python
>>> from shutil import copy, copytree
>>> import sima.misc

>>> copytree(sima.misc.example_data(), 'example.sima')
>>> copy(sima.misc.example_tiff(), 'example.tif')
>>> copy(sima.misc.example_tiff(), 'example_Ch1.tif')
>>> copy(sima.misc.example_tiff(), 'example_Ch2.tif')
>>> copy(sima.misc.example_hdf5(), 'example.h5')
>>> copy(sima.misc.example_imagej_rois(), 'imageJ_ROIs.zip')
```

2.2.3 Creating an ImagingDataset object

The SIMA package is centers around the ImagingDataset object class. A single ImagingDataset object can contain imaging data from multiple sequences (i.e. continuous imaging epochs/trials) acquired at the same imaging location during the same imaging session. The subsections below provide examples of how to initialize ImagingDataset objects using raw data in a variety of formats, including Numpy arrays, TIFF files, and HDF5 files.

The ImagingDataset object is permanently stored in the location (ending with extension .sima) specified during initialization. Results of segmentation, signal extraction, and other alterations to the ImagingDataset object are automatically saved to this location.

Numpy arrays

To begin with, we create some Numpy arrays containing random data. The shape of these arrays is (num_frames, num_planes, num_rows, num_columns, num_channels).
>>> import numpy as np
>>> cycle1_channel1 = np.random.rand(100, 1, 128, 128, 1)
>>> cycle1_channel2 = np.random.rand(100, 1, 128, 128, 1)
>>> cycle2_channel1 = np.random.rand(100, 1, 128, 128, 1)
>>> cycle2_channel2 = np.random.rand(100, 1, 128, 128, 1)

Once we have the Numpy arrays containing the imaging data, we create the ImagingDataset object as follows.

```python
>>> sequences = [sima.Sequence.join(
...    sima.Sequence.create('ndarray', cycle1_channel1),
...    sima.Sequence.create('ndarray', cycle1_channel2)),
...    sima.Sequence.join(
...    sima.Sequence.create('ndarray', cycle2_channel1),
...    sima.Sequence.create('ndarray', cycle2_channel2))
```  
```python
>>> dataset = sima.ImagingDataset(
...   sequences, 'example_np.sima', channel_names=['green', 'red'])
```  
### Multipage TIFF files

For simplicity, we consider the case of only a single cycle and channel.

```python
>>> sequences = [sima.Sequence.create('TIFF', 'example_Ch1.tif')
```  
```python
>>> dataset = sima.ImagingDataset(sequences, 'example_TIFF.sima')
```  
### HDF5 files

The argument 'yxt' specifies that the first index of the HDF5 array corresponds to the row, the second to the column, and the third to the time.

```python
>>> sequences = [sima.Sequence.create('HDF5', 'example.h5', 'yxt')
```  
```python
>>> dataset = sima.ImagingDataset(sequences, 'example_HDF5.sima')
```  
#### 2.2.4 Loading ImagingDataset objects

A dataset object can also be loaded from a saved path with the .sima extension.

```python
>>> dataset = sima.ImagingDataset.load('example.sima')
```  
#### 2.2.5 Motion correction

The SIMA package implements a variety of approaches for motion correction. To use one of these approaches, the user first creates an object encapsulating the approach and a choice of parameters. For example, an object encapsulating the approach of translating imaging planes in two dimensions with a maximum displacement of 15 rows and 30 columns can be created as follows:

```python
>>> import sima.motion
>>> mc_approach = sima.motion.PlaneTranslation2D(max_displacement=[15, 30])
```  
```python
>>> sequences = [sima.Sequence.create('TIFF', 'example_Ch1.tif')
```  
```python
>>> dataset = mc_approach.correct(sequences, 'example_translation2D.sima')
```
In the following example, the SIMA package is used for motion correction based on a hidden Markov model (HMM). The `sima.motion.HiddenMarkov2D` class takes the initialize arguments to specify its parameters (e.g., an optional argument is used to indicate that the maximum possible displacement is 20 rows and 30 columns). The `correct()` method takes the same arguments as are used to initialize an imaging dataset object, as well as some additional optional arguments.

```python
>>> mc_approach = sima.motion.HiddenMarkov2D(
...     granularity='row', max_displacement=[20, 30], verbose=False)
>>> dataset = mc_approach.correct(sequences, 'example_mc.sima')
```

When the signal is of interest is very sparse or highly dynamic, it is sometimes helpful to use a second static channel to estimate the displacements for motion correction. The example below is for the case where the first channel contains a dynamic GCaMP signal whose large variations would confuse the motion correction algorithm, and the second channel contains a static tdTomato signal that provides a stable reference.

```python
>>> sequences = [
...     sima.Sequence.join(sima.Sequence.create('TIFF', 'example_Ch1.tif'),
...     sima.Sequence.create('TIFF', 'example_Ch1.tif'))
... ]
>>> dataset = mc_approach.correct(
...     sequences, 'example_mc2.sima', channel_names=['GCaMP', 'tdTomato'],
...     correction_channels=['tdTomato'])
```

When motion correction is invoked as above, only the tdTomato channel is used for estimating the displacements, which are then applied to both channels.

### 2.2.6 Segmentation and ROIs

#### Automated segmentation

The SIMA package implements a number of approaches for automated segmentation. To use one of the approaches, the first step is to create an object encapsulating the approach and a choice of parameters. For example, an object representing the approach of segmenting a single-plane dataset with spatiotemporal independent component analysis (STICA) can be created as follows:

```python
>>> import sima.segment
>>> stica_approach = sima.segment.STICA(components=5)
```

We can also add post-processing steps to this approach, for example to convert the STICA masks into sparse regions of interest (ROIs), smooth their boundaries, and merge overlapping ROIs.

```python
>>> stica_approach.append(sima.segment.SparseROIsFromMasks())
>>> stica_approach.append(sima.segment.SmoothROIBoundaries())
>>> stica_approach.append(sima.segment.MergeOverlapping(threshold=0.5))
```

Once the approach has been created, it can be passed as an argument to the `segment()` method of an `ImagingDataset`. The `segment()` method can also take an optional label argument for the resulting set of ROIs.

```python
>>> dataset = sima.ImagingDataset.load('example.sima')
>>> rois = dataset.segment(stica_approach, 'auto_ROIs')
```

#### Editing, creating, and registering ROIs with ROI Buddy

Note that the `ImagingDataset` object can be loaded with the ROI Buddy graphical user interface (GUI) for manual editing of existing the ROI lists, creation of new ROI lists, or registration of ROI lists across multiple experiments.
in which the same field of view is imaged. For more details, consult the ROI Buddy documentation.

**Importing ROIs from ImageJ**

ROIs can also be imported from ImageJ, as shown in the following example.

```python
>>> from sima.ROI import ROIList

>>> dataset = sima.ImagingDataset.load('example.sima')

>>> rois = ROIList.load('imageJ_ROIs.zip', fmt='ImageJ')

>>> dataset.add_ROIs(rois, 'from_imageJ')

>>> dataset.ROIs.keys()

['from_imageJ', 'auto_ROIs']
```

**Mapping ROIs between datasets**

Sometimes, for example when imaging the same field of view over multiple days, one wishes to segment the same structures in separate ImagingDataset objects. If all of the ImagingDataset objects have been segmented, then the results of the segmentations can be registered with the ROI Buddy GUI as mentioned previously. If, however, only one of the datasets has been segmented, the results of the segmentation can be applied to the other datasets by applying to each ROI the affine transformation necessary to map one imaged field of view onto the other. This can be done either with the ROI Buddy GUI or with a call to the `import_transformed_ROIs()` method, whose arguments allow for specification of the channels used to align the two datasets, the label of the `obj:ROIList` to be transformed from one dataset to the other, the label that will be applied to the new `ROIList`, and whether to copy the properties of the ROIs as well as their shapes.

```python
>>> source_dataset = sima.ImagingDataset.load('example.sima')

>>> target_dataset = sima.ImagingDataset.load('example_mc2.sima')

>>> target_dataset.ROIs.keys()

[]

>>> target_dataset.import_transformed_ROIs(
...    source_dataset, source_channel='green', target_channel='GCaMP',
...    source_label='from_imageJ', target_label='transformed',
...    copy_properties='True')

>>> target_dataset.ROIs.keys()

['transformed']
```

This approach allows the user to focus on careful manual curation of the segmentation for a single ImagingDataset, with the results of this segmentation then applied to all datasets acquired at the same field of view.

**Accessing stored ROIs**

Whenever ROIs are created or imported, they are permanently stored as part of the ImagingDataset object. The ROIs can be recovered at any time using the label specified at the time when the ROIs were created.

```python
>>> dataset = sima.ImagingDataset.load('example.sima')

>>> dataset.ROIs.keys()

['from_imageJ', 'auto_ROIs']

>>> rois = dataset.ROIs['auto_ROIs']
```

### 2.2.7 Extraction

Once the ROIs have been edited and registered, the dataset can be loaded, and then dynamic fluorescence signals can be extracted from the ROIs with the `extract()` method.
The extracted signals are permanently saved with the `ImagingDataset` object and can be accessed at any time with the command `signals()` method.

```python
>>> dataset = sima.ImagingDataset.load('example.sima')
```

### 2.2.8 Exporting data

Data can be exported from the SIMA `ImagingDataset` objects at various stages of the analysis. This allows SIMA to be used for early stages of data analysis, and then for the exported data to be analyzed with separate software. If, however, further analysis is to be performed with Python, such exporting may not be necessary. The subsections below contain examples showing how to export image data and signal data.

#### Image data

The `ImagingDataset` class has two methods for exporting image data, `export_frames()` and `export_averages()`, which export either all the frames or the time averages of each channel, respectively. These methods can be used to view the results of motion correction, as shown in the following example.

```python
>>> import sima.motion
>>> sequences = [sima.Sequence.create('TIFF', 'example_Ch1.tif')]
>>> dataset = mc_approach.correct(sequences, 'example_mc3.sima')
>>> dataset.export_averages(['avgs.tif'], fmt='TIFF16')
>>> dataset.export_frames([['frames.tif']], fmt='TIFF16')
```

The paths to which the exported data are saved are organized as a list with one filename per channel for the `export_averages()` method, or as a list of lists (organized analogously to the sequence used to initialize an `ImagingDataset` object) for the `export_frames()` method. If however, the export format is specified to HDF5, then the filenames for `export_frames()` should be organized into a list with one filename per cycle, since both channels are combined into a single HDF5 file.

```python
>>> dataset.export_frames(['exported_frames.h5'], fmt='HDF5')
```

#### Signal data

For users wishing to analyze the extracted signals with an external program, these signals can be exported to a CSV file.

```python
>>> dataset = sima.ImagingDataset.load('example.sima')
>>> dataset.export_signals('example_signals.csv', channel='green')
```

The resulting CSV file contains the `id`, `label`, and `tags` for each ROI, and the extracted signal from each ROI at each frame time.
2.2.9 Complete example

Below are the contents of workflow.py in the examples directory provided with the SIMA source code.

```python
#!/usr/bin/env python

"""
This file provides a demonstration of how to use the SIMA package.

In order to run this file, you will need to download the file
http://www.losonczylab.org/workflow_data.zip and extract it in your
current working directory.

"""
from __future__ import 
print_function
from builtins import input
from builtins import range

import sima
import sima.motion
import sima.segment

import sima
import sima.motion
import sima.segment

import sima
import sima.motion
import sima.segment

# Generate the filenames with Python list comprehensions.
tiff_filenames = [
    ['workflow_data/Cycle{n1:02d}_Ch{n2}.tif'.format(n1=cycle, n2=channel)
     for channel in range(1, 3)
     for cycle in range(1, 16)
]

# The resulting filenames are printed for clarification.
print("TIFF filenames:

   ", tiff_filenames)

# Finally, we construct a MultiPageTIFF iterable using each of the filenames.
sequences = [
    sima.Sequence.join(*[sima.Sequence.create('TIFF', chan) for chan in cycle])
    for cycle in tiff_filenames
]

dataset_path = 'workflow_data/dataset.sima'
correction_approach = sima.motion.HiddenMarkov2D(num_states_retained=30,
                                                 max_displacement=[20, 30])
dataset = correction_approach.correct(  
```
sequences, dataset_path, channel_names=['tdTomato', 'GCaMP'],
trim_criterion=0.95)

# Export the time averages for a manuscript figure.
dataset.export_averages(['workflow_data/tdTomato.tif',
'workflow_data/GCaMP.tif'])

# Generate the output filenames with Python list comprehensions.
output_filenames = [
    [channel.replace('.tif', '_corrected.tif') for channel in cycle]
for cycle in tiff_filenames]

# The resulting filenames are printed for clarification.
print("Output filenames:

", output_filenames)

# Export the corrected frames for a presentation.
dataset.export_frames(output_filenames, fill_gaps=True)

# At this point, one may wish to inspect the exported image data to evaluate
# the quality of the motion correction before continuing.
while True:
    input_ = input("Continue? (y/n): ")
    if input_ == 'n':
        exit()
    elif input_ == 'y':
        break

# Segment the field of view into ROIs using the method for CA1 pyramidal cells
# and parameters that were determined based on the imaging magnification.
segmentation_approach = sima.segment.PlaneCA1PC(
    channel='GCaMP',
    num_pcs=30,
    max_dist=(3, 6),
    spatial_decay=(3, 6),
    cut_max_pen=0.10,
    cut_min_size=50,
    cut_max_size=150,
    x_diameter=14,
    y_diameter=7,
    circularity_threshold=.5,
    min_roi_size=20,
    min_cut_size=40)

dataset.segment(segmentation_approach, 'auto_ROIs')

# At this point, one may wish to edit the automatically segmented ROIs using
# the ROI Buddy GUI before performing signal extraction.
while True:
    input_ = input("Continue? (y/n): ")
    if input_ == 'n':
        exit()
exit()

elif input_ == 'y':
    break

##############################################################################
# #
# PART 4: Extracting fluorescence signals from the ROIs. #
# #
# Reload the dataset in case any changes have been made with ROI Buddy
dataset = sima.ImagingDataset.load(dataset_path)

# Extract the signals. By default, the most recently created ROIs are used.
dataset.extract(signal_channel='GCaMP', label='GCaMP_signals')

# Export the extracted signals to a CSV file.
dataset.export_signals('example_signals.csv', channel='GCaMP',
                        signals_label='GCaMP_signals')

##############################################################################
# #
# PART 5: Visualizing data using Python. #
# #
# #
# import necessary functions from matplotlib
from matplotlib.pyplot import plot, show

# plot the signal from an ROI object, with a different color for each cycle
raw_signals = dataset.signals('GCaMP')['GCaMP_signals']['raw']
for sequence in range(3):
    # plot data from the first 3 cycles
    plot(raw_signals[sequence][3])  # plot the data from ROI #3
show(block=True)

2.3 SIMA API

The public API of the SIMA package is described below. Functionality not described in this documentation may not remain stable across versions and should therefore not be used.

2.3.1 Sequence objects

2.3.2 ImagingDataset objects

2.3.3 Motion correction

The SIMA package can be used to motion correct sequentially acquired images. The package implements several strategies, all of which have a common interface described below.

The specific strategies for motion correction are listed below. These strategies must be initialized as documented below, and then can be applied to datasets using the generic interface described above.
The SIMA package implements multiple segmentation strategies, which share a common interface defined by the abstract class `SegmentationStrategy`. Any strategy with this interface can be used in conjunction with the `ImagingDataset.segment()` method to segment an `ImagingDataset` object.

### Segmentation strategies

**Note:** Please consider contributing additional methods to the SIMA project.

The specific segmentation strategies that have been implemented are documented below. Once initialized as documented, these strategies all share the above interface.

#### Plane-Wise Segmentation

#### Spatialtemporal Independent Component Analysis

#### Normalized cuts

#### Affinity Matrix Methods

#### CA1 pyramidal cells

#### Post-Processing Steps

Any number of post-processing steps can be added to a segmentation method using the `SegmentationStrategy.append()` method. These appended post-processing steps must have the interface defined by `PostProcessingStep` below. The appended steps can be selected from those documented in this section, or can be created by the user by subclassing any of the classes listed below.
2.3.5 Regions of Interest (ROIs)

ROI class

ROIList class

2.3.6 Signal extraction

Details

2.4 ROI Buddy GUI

Contents

- ROI Buddy GUI
  - Introduction
    * Typical Workflow
  - Installation
    * Windows
    * Mac OS X
  - User Interface
    * File Menu
    * Control Panel
      - Toggling modes
      - Initializing SIMA imaging dataset and ROI List objects
      - Registering ROI objects across multiple imaging sessions
      - Toggling the visibility of ROIs
  - Keyboard shortcuts

2.4.1 Introduction

The ROI Buddy GUI can be used for viewing, creating, editing, and tagging the regions of interest (ROIs) associated with multiple SIMA imaging datasets simultaneously. In addition, it can be used for registering ROIs from different imaging sessions of the same field-of-view, allowing for the longitudinal tracking of cells in serial imaging experiments.

Typical Workflow

- Load multiple SIMA imaging datasets corresponding to different recordings of the same field of view.
- For each dataset, select the channel you wish to edit and align.
- Create or load pre-drawn ROIs for each dataset, editing the ROIs as necessary in Edit Mode.
- Tag cells in one dataset based on some criteria, e.g., morphology.
- Enter Align mode, bringing all the ROIs into alignment.
- Choose “Register ROIs” to assign a common ID property to overlapping ROIs.
- Use the merge and unmerge tools to manually fix any incorrect groupings.
• Once registration is satisfactory, choose “Propagate Tags” to assign the morphology tags to all matched ROIs across imaging sessions.

• Save ROIs.

2.4.2 Installation

The ROI Buddy GUI has the following dependencies which should be installed prior to installation of ROI Buddy:

• sima>=1.0 (https://pypi.python.org/pypi/sima)
• PyQt4 (http://www.riverbankcomputing.co.uk/software/pyqt)
• guidata (https://code.google.com/p/guidata/)
• guiqwt (https://code.google.com/p/guiqwt/)

Once these dependencies are installed, the ROI Buddy can be installed from the Python Package Index using pip:

\$ pip install roibuddy --user

Alternatively, the ROI Buddy GUI can be built from source. Source code for released versions can be downloaded from the Python Package Index, and code under development can be downloaded from the project’s GitHub page.

Installation of ROI Buddy will create an executable file that can be run by typing “roibuddy” into your terminal. If this command does not work, check the output of the installation command to determine the directory where the executable was placed, and either add this directory to your path or run the executable directly from that location.

Windows

After installing SIMA, we recommend installing pre-compiled wheels from (http://www.lfd.uci.edu/~gohlke/pythonlibs/) for PyQt4, PyQt4, guidata, and guiqwt (in that order), before installing ROI Buddy from the Python Package Index.

Mac OS X

We recommend using MacPorts for installing the dependencies. After installing SIMA using MacPorts as described here, run the following command in Terminal to install an additional dependency:

\$ sudo port install py27-pyqwt

Then download and install guidata and guiqwt before installing ROI Buddy from the Python Package Index.
## 2.4.3 User Interface

### File Menu

<table>
<thead>
<tr>
<th>File Menu Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add T-Series</td>
<td>Adds a .sima directory to the list of imaging datasets</td>
</tr>
<tr>
<td>Auto add t-series</td>
<td>Recursively walk the selected directory and add directories ending in .sima. Optionally a tag argument can be specified, in which case tag must be contained in the .sima directory name</td>
</tr>
<tr>
<td>Save current ROIs</td>
<td>Save the active ROIList associated with the active imaging dataset</td>
</tr>
<tr>
<td>Save all ROIs</td>
<td>For each loaded imaging dataset, save the active ROIList with the same name</td>
</tr>
<tr>
<td>Edit Label</td>
<td>Edit the label property associated with the selected ROI objects. If two polygons in the active ROI List share the same label they are assumed to be disjoint regions comprising a single ROI object.</td>
</tr>
<tr>
<td>Add Tags</td>
<td>Add comma-delimited strings to the ROI tags property. This feature is useful for tagging ROIs on morphology, genetic identity, etc. and filtering based on this criteria in subsequent analysis</td>
</tr>
<tr>
<td>Remove Tags</td>
<td>Clear the tags associated with the selected ROIs</td>
</tr>
<tr>
<td>Edit Tags</td>
<td>Manually edit the list of comma-delimited tag strings associated with the selected ROIs</td>
</tr>
<tr>
<td>Merge</td>
<td>• Edit Mode: If the selected polygons are contiguous, merge them into a single polygon defined by the convex hull containing the selected polygons. If regions are discontinuous, assign them a common label property, indicating that these disjoint structures comprise a single ROI upon saving.</td>
</tr>
<tr>
<td></td>
<td>• Align Mode: Select exactly two polygons from different imaging datasets. The selected regions will be assigned the same id property, indicating that they are the same structure imaged during different sessions. The id assigned corresponds to that of the ROI belonging to the largest cluster.</td>
</tr>
<tr>
<td>Unmerge</td>
<td>Unmerge the selected ROI from any groupings. In both Edit and Align modes, the id property is cleared. In Edit mode, the label property is set to a unique value.</td>
</tr>
<tr>
<td>Quit</td>
<td>Quit ROI Buddy application</td>
</tr>
</tbody>
</table>
Control Panel

Toggling modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit</td>
<td>Toggling to edit mode allows for editing, creation, and deletion of ROIs. In addition, this is the mode in which ROI tagging and labeling takes place.</td>
</tr>
<tr>
<td>Align</td>
<td>Toggling to align mode allows for the transformation of ROI Lists from separate imaging sessions into a common space, allowing for simultaneous visualization of independently drawn sets. In this view, ROI objects can be deleted (as in the case of singleton ROIs that were perhaps a false positive), and ROIs can be registered across days.</td>
</tr>
</tbody>
</table>

Initializing SIMA imaging dataset and ROI List objects

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove / Add</td>
<td>Add or remove SIMA datasets from the dataset list.</td>
</tr>
<tr>
<td>ROI Set</td>
<td>A dropdown allowing for selection of the ROI List currently being modified. These ROI Lists correspond to the keys of imagingDataset.ROIs(). By default, the most recently modified ROI List is displayed upon initialization.</td>
</tr>
<tr>
<td>New / Delete Set</td>
<td>Create or delete ROI Lists for the active SIMA imaging dataset. These lists are stored in the associated imagingDatset.ROIs dictionary.</td>
</tr>
</tbody>
</table>

Registering ROI objects across multiple imaging sessions

If multiple datasets have been loaded from the same field of view, it is possible to align the ROIs and commonly identify them so that they can be tracked across sessions. The currently selected dataset will act as a template to which all other datasets will be aligned. Each dataset is automatically aligned to the template dataset by calculating an affine transformation between time averaged images that produces maximal similarity. A clustering algorithm based on the Jaccard Index is used to match cells between datasets, which can be manually adjusted by merging/unmerging ROIs from the automatic clusters. Once registered, ROIs that are matched across days are assigned the same id property which is denoted visibly by giving them all the same color.
Register ROIs | Begin automatic registration using the currently active dataset as the registration template. If datasets do not match, an error dialog will identify the datasets that could be aligned.

Propagate Tags | In align mode, the propagate tags button distributes tags across ROI objects from different imaging datasets sharing the same id property. The user can select which tags to propagate. Thus, regions need only be tagged in one dataset, and these tags can be added to other recordings of the same region.

**Warning:** In align mode, it is necessary that all imaging datasets loaded must be roughly of the same field of view. Alignment is based upon an affine transformation with 6 degrees of freedom. If a transform between time averaged images cannot be calculated, an error message will be displayed printing the directories of the incompatible sets.

**Toggling the visibility of ROIs**

| Show ROIs / All | Toggle the visibility of ROI polygons. In align mode, Show All results in the transformation of all ROIs into a common space, allowing for visual inspection of alignment across imaging sessions.

**2.4.4 Keyboard shortcuts**

- **m (edit mode)** merge separate ROIs into a single ROI object
- **m (align mode)** merge selected ROIs into the same cluster, assigning them the same ID attribute
- **u (align mode)** unmerge ROI from its cluster and assign it a unique ID attribute
- **f (edit mode)** select freeform tool
- **s (edit mode)** select pointer-selection tool
- **d** delete
- **r** randomize ROI colors

**2.5 Publications**

**2.5.1 Publications about SIMA**

If you use SIMA for your research, please cite the following paper in any resulting publications:


**2.5.2 Research using SIMA**


Please email software@losonczylab.org to let us know of any publications that are missing from the above list.

2.6 Credits

2.6.1 Development

• Patrick Kaifosh: initial development
• Jeffrey Zaremba: initial development
• Nathan Danielson: initial development
• Jack Bowler: test suite, stICA segmentation
• Tim Machado: spike inference
• Christoph Schmidt-Hieber: parallelized post-processing, DFT motion correction
• Lloyd Russell: DFT motion correction

2.6.2 Testing / Bug Fixes

• Ben Shababo
• Clay Lacefield
• Matthew Lovett-Barron
• Matthew J. Davis
• Thomas Hainmüller
• Ran Rubin
• Gokhan Ordek
• Nathan Kern
• Masatoshi Kasai
• Yoshiyuki Yamada
• Maximilian Hoffmann
• Erik Flister

2.6.3 External code

• tiffile.py from Christoph Gohlke
• NIH ImageJ ROI parsing from Luis Pedro Coelho
2.6.4 Funding

- Patrick Kaifosh is a Howard Hughes Medical Institute International Student Research fellow.

2.7 How you can help

2.7.1 Contact us

If you are using SIMA, please consider emailing software@losonczylab.org to let us know. Knowledge of our user base is extremely helpful for planning new features, attracting volunteers, and fundraising. Words of thanks are also very encouraging to the development team.

2.7.2 Spread the word

Please pass along information about SIMA to people who would be interested in using it or contributing to its development.

2.7.3 Improve the software

We welcome contributions to the SIMA package in many forms.

Bug reports

Please report bugs to the SIMA Issue Tracker. When doing so, please adhere to the following guidelines:

- Search the issues to make sure the problem has not already been addressed.
- Describe the problem in detail.
- Include a short example (e.g. a few lines) that reproduces the problem.

If you are not sure whether your problem is a bug, or simply a question about how to use SIMA, please post your question to the User Support section of the SIMA forum.

Feature requests

If you think that a new feature would make SIMA more helpful for your research, please add an issue to the Issue Tracker. When doing so, please follow the

- Search to see if the feature has already been requested. If so, please add a comment to that issue. This comment can include additional information about your requirements or just help to give a sense of how many people desire a certain feature.
- If the feature has not already been requested, please create a new issue with a “feature request” tag. Please be specific as possible about your requirements.

Documentation

We appreciate improvements to any parts of the documentation that are unclear, incomplete, or incorrect. Please see the Development instructions for more details on how to contribute to the documentation.
Development

To contribute to the source code, please follow the Fork & Pull model. We have not developed specific guidelines for SIMA, so please follow the principles outlined in SciPy’s contributing guidelines.

The efforts of SIMA contributors are coordinated through the Issue Tracker. Also, feel free to post any questions about contributing to the User Support section of the SIMA forum.
Citing SIMA

If you use SIMA for your research, please cite the following paper in any resulting publications:

Indices and tables

- genindex
- modindex
- search