SOFTWARE TOOL ARTICLE

flimview : A software framework to handle, visualize and analyze FLIM data [version 1; peer review: 1 approved, 1 approved with reservations]

Matias Carrasco Kind¹, Mantas Zurauskas^{2,3}, Aneesh Alex^{2,4},
Marina Marjanovic^{2,3,5,6}, Prabuddha Mukherjee^{1,2}, Minh Doan⁴,
Darold R. Spillman Jr.^{2,3}, Steve Hood^{2,7}, Stephen A. Boppart ^{2,3,5,6,8}

¹National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA

²GSK Center for Optical Molecular Imaging, University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA

³Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA

⁴GlaxoSmithKline, Collegeville, PA, USA

⁵Department of Bioengineering, University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA

⁶Carle Illinois College of Medicine, University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA

⁷GlaxoSmithKline, Stevenage, UK

⁸Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA

V1 First published: 08 Jun 2020, 9:574
<https://doi.org/10.12688/f1000research.24006.1>

Latest published: 08 Jun 2020, 9:574
<https://doi.org/10.12688/f1000research.24006.1>

Abstract

flimview is a bio-imaging Python software package to read, explore, manage and visualize Fluorescence-Lifetime Imaging Microscopy (FLIM) images. It can open the standard FLIM data file conventions (e.g., sdt and ptu) and processes them from the raw format to a more readable and manageable binned and fitted format. It allows customized kernels for binning the data as well as user defined masking operations for pre-processing the images. It also allows customized fluorescence decay fitting functions and preserves all of the metadata generated for provenance and reproducibility. Outcomes from the analysis are lossless compressed and stored in an efficient way providing the necessary open-source tools to access and explore the data. flimview is open source and includes example data, example [Jupyter](#) notebooks and tutorial documentation. The package, test data and documentation are available on [Github](#).

Keywords

FLIM, bio-imaging, microscopy, visualization, fluorescence, Python

This article is included in the [Python](#) collection.

Open Peer Review

Approval Status

	1	2
version 1		
08 Jun 2020	view	view
1. Xavier Michalet , University of California, Los Angeles (UCLA), Los Angeles, USA		
2. Nirmal Mazumder , Manipal School of Life Sciences, Manipal Academy of Higher Education (MAHE), Manipal, India		
Any reports and responses or comments on the article can be found at the end of the article.		

Corresponding author: Matias Carrasco Kind (mcarras2@illinois.edu)

Author roles: Carrasco Kind M: Conceptualization, Software, Visualization, Writing – Review & Editing; Zurauskas M: Conceptualization, Methodology, Validation, Writing – Review & Editing; Alex A: Conceptualization, Project Administration, Writing – Review & Editing; Marjanovic M: Conceptualization, Writing – Review & Editing; Mukherjee P: Conceptualization, Methodology, Writing – Review & Editing; Doan M: Conceptualization, Supervision; Spillman Jr. DR: Resources, Supervision; Hood S: Supervision; Boppert SA: Supervision

Competing interests: S.A.B receives research grant support from GlaxoSmithKline related to the research described here, and reports receiving consulting fees from and owning equity interest in PhotoniCare, Inc., Diagnostic Photonics, Inc. and LiveBx, LLC. A.A., M.D. and S.H. are employees and shareholders of GlaxoSmithKline. M.Z, M.M., P.M., D.R.S. declare no potential conflict of interest.

Grant information: The work presented here was funded by GlaxoSmithKline.

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2020 Carrasco Kind M *et al.* This is an open access article distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Carrasco Kind M, Zurauskas M, Alex A *et al.* *flimview : A software framework to handle, visualize and analyze FLIM data [version 1; peer review: 1 approved, 1 approved with reservations]* F1000Research 2020, 9:574 <https://doi.org/10.12688/f1000research.24006.1>

First published: 08 Jun 2020, 9:574 <https://doi.org/10.12688/f1000research.24006.1>

Introduction

Fluorescence lifetime imaging microscopy (FLIM) is an imaging technique where the image contrast is derived from the differences in the exponential decay rate of the fluorescence from a fluorescent sample (Bower *et al.*, 2018).

In modern two-photon FLIM, the most widely used detection method is known as time-correlated single-photon counting (TCSPC) (Becker *et al.*, 2004). TCPC employs a pulsed excitation source, such as a laser or a light emitting diode and measures the timing of arrival of single photons originating from the fluorescent sample on a detector to reconstruct the fluorescence lifetime decay (McGinty *et al.*, 2016). Typically, different fluorophores can be characterized by their fluorescence lifetimes. Furthermore, fluorescence lifetimes of a fluorophore can be affected by its environment, therefore FLIM have emerged as a valuable tool, providing unique contrast mechanisms for biomedical imaging (Bower *et al.*, 2019; Hirvonen & Suhling, 2016; Ranawat *et al.*, 2019).

To reconstruct TCSPC based FLIM image, several non-trivial data processing steps are necessary. For extracting the lifetime and amplitude data from initially saved raw files, first the data has to be imported from proprietary files such as Becker and Hickl `.sdt`, or PicoQuant `.ptu`. Next, signal preprocessing through denoising or binning has to be applied to ensure robust performance in low photon count scenarios. Finally, after compensating for instrument response function, biexponential decay curves are fitted to extract lifetime and amplitude values for two dominant components at each pixel.

Several commercial and open source packages are already available for FLIM analysis (Bergmann, 2003; Fli; Warren *et al.*, 2013). However, in practice, when custom FLIM analysis capabilities are required, this can lead to a need for building complicated signal processing and analysis pipelines (Borhani *et al.*, 2019; Bower *et al.*, 2017; Cao *et al.*, 2020), involving a mix of commercial tools, open-source software and bespoke analysis algorithms. While other packages have been published, (Ballesteros *et al.*, 2019), we expect that `flimview`, with its focus on simplicity, and versatility, will serve as a useful tool for the FLIM community which can also contribute to its growth.

Here we present a Python package, `flimview`, which provides a user-friendly toolkit for opening, fitting and visualizing FLIM data. The package can be used as a stand-alone tool for analyzing and visualizing FLIM data, or it can be used as a basis for creating end-to-end streamlined analytical pipelines in Python.

Methods

Implementation

`flimview` is a library module implemented in Python 3 and can be installed using PyPI (e.g., `pip install flimview`). Installation instructions using Conda are also included within the repository. A complete list of requirements can be found at the package Github page but include standard Python libraries such as `pandas`, `scipy`, `matplotlib`, `h5py`, etc. The code is released under the open-source NCSA license which is based on the MIT open source license, allowing full and free use for both commercial and non-commercial purposes. Full usage documentation, Jupyter notebooks (web-browser interface that allows live Python codes), test data and others, including guidelines for contributions and issue reporting, are included in the project repository. This package has been tried in Windows, Mac OS and Linux Operating Systems. In general `flimview` works better in the latter two for which it is also expected to get supported development.

Operation

`flimview` consists of several utility libraries for storage, visualization and manipulation of FLIM data. After reading and processing the raw data (`.ptu` or `.sdt` files), the input data is stored in the main class of the package, a `FlimCube`, which is an object that represents 3D data (2D spatial dimensions and one temporal dimension) allowing multiple methods to access and manipulate the data cube.

Among its attributes is the header which includes all the metadata available for the dataset, sizes of the arrays, resolution and whether this cube is binned and/or masked (see Binning and Masking sections below). When the data cube is masked, a process described in the following sections, the pixel-level mask is also included as an attribute in the `FlimCube` object. Due to its construction, we consider a `FlimCube` to be self-explanatory as it contains all needed information (including its corresponding metadata) to analyze, visualize and fit the data.

Binning. `flimview` includes a binning function with several pre-defined kernels, although a user-defined kernel is also possible and can be easily added. The binning procedure is simply a convolution kernel applied to the

image to increase the signal or to enhance its features. This function takes a `FlimCube` as an input and returns a `FlimCube` as output by copying all of the metadata and properties from the input data. The following snippet shows an example on how to bin an existing `FlimCube` using a Gaussian kernel of size 9x9 and a sigma of 3 (the size of the kernel is $2 * b + 1$, where b is the bin size)

```
import flimview.flim as flim
import flimview.io_utils as io # IO utils

data, header = io.read_sdt_file (sdtfile)
FC = flim.FlimCube (data, header) # FlimCube Class
FCbinned = flim.binCube (FC, bin=4, kernel='gauss', sigma=3)
# FCbinned is also a FlimCube object
#
```

The kernel functions included in `flimview` are Gaussian kernel, Airy disk kernel, linear kernel, and a flat kernel. Customized kernels can be easily incorporated. [Figure 1](#) shows examples of different 9x9 kernels where we can observe how the weight is distributed around the central pixel. In the case of a Gaussian kernel or an Airy disk kernel, the width of the kernel can be customized using the sigma σ parameter. For reference, the Gaussian kernel is given by:

$$I(x) = I_0 e^{-x^2/2\sigma^2}$$

where x is the distance to the center pixel and I_0 is a normalization factor (usually just a unit). In the case of the Airy disk, the kernel is given by:

$$I(x) = I_0 \left[\frac{2J_1(x)}{x} \right]^2$$

Figure 1. Example of different 9x9 kernels used for binning.

where $J_1(x)$ is the Bessel function of the first kind of order one. In both cases, $I(x)$ is discretized and normalized to the unit across the 2d binning window.

Masking. FlimCube also includes methods for looking at the header information and for masking pixels below a given integrated intensity, or below a given peak threshold in the time series, or even for a custom made geometry mask. The pixels masked this way will not be used during the fitting analysis while the mask is also saved within the same input FlimCube object. The following snippet shows how to apply a mask for a given FlimCube :

```
import flimview.flim as flim
import flimview.io_utils as io #IO utils

# Read data
# sdtfile = ...
data, header = io.read_sdt_file(sdtfile)
FC = flim.FlimCube(data, header)
# To mask by intensity
FC.mask_intensity(100)
# To mask by peak
FC.mask_peak(5)
# To mask by a given geometry
FC.mask_peak(0, mask=custom_numpy_masked_array)
#
```

Figure 2 shows different masks applied to the example raw (top) and binned (bottom) data. Masks can be combined and can be transferred between binned and raw data.

Fitting. Once the data is read and binned, and have the low signal and error pixels masked, flimview provides a function to fit a decay function for every pixel in the temporal data using a predefined model (from the module `models.py` which can be customized and also parallelized. By default a double exponential is used as follows:

$$f(t) = a_1 e^{-t/\tau_1} + a_2 e^{-t/\tau_2} + l_0$$

where a_1 and a_2 are the amplitudes for the given exponential factor. Usually, there is an extra constraint given by $a_1 + a_2 = 1$. τ_1 and τ_2 are the mean lifetime for each exponential. l_0 is a level constant.

Before fitting the entire cube, a single fit is done to the mean intensity values to obtain initial guess parameters for the fitting procedures as shown in Figure 3, as well as the boundaries to each parameter which can also be provided by hand. The snippet below shows how this is done using flimview :

Figure 2. Different Masking approaches to the raw data (top) and binned data (bottom). Masks can be combined. Masked pixels will not be used in the analysis.

Figure 3. Example of fitting the average and normalized pixel decay to determine initial parameters.

```
import flimview.flim as flim
# Assuming a FlimCube FC, let's compute the average of the pixel values
timesteps, mean_pixel = flim.meanDecay (FCbin)
# Let's clean these by a threshold, normalize and shift to the maximum
timesteps_clean, mean_pixel_clean, max_value, time_shift = flim.cleanCurve (timesteps,
    mean_pixel, norm=True, threshold=0.02)
# Let's assume a model (double exponential with a level parameter)
mymodel = models.model1
# Let's fit that mean pixel using fitPixel function
# xf, yf are the time and value used for the fitting, pfit is the fitted values, pcov
# is the parameters covariance matrix and chi2
# is the Chi - square value of the fit
xf, yf, pfit, pcov, chi2 = flim.fitPixel(
    timesteps,
    mean_pixel,
    mymodel,
    initial_p=[0.8, 0.3, 3, 0.03],
    bounds=(0, np.inf),
    norm = True)
#
```

Note that the data is normalized, cleaned, and shifted with respect its peak. For example, to generate [Figure 3](#) we can use the following:

```
# We can plot the fitted data (only considering from the maximum value)
plt.plot(timesteps - time_shift, mean_pixel/max_value, '.', label='original data')
plt.plot(xf, yf, '.', label='fitted data')
plt.plot(xf, mymodel (xf, *pfit))
plt.xlabel('time [ns]', fontsize=15)
plt.ylabel('Intensity', fontsize=15)
plt.text(6, 0.5, flim.printModel (mymodel, pfit, pcov, chi2, oneliner=False))
plt.legend(loc=0)
```

After those parameters are defined and a model is selected a function `fitCube` takes a `FlimCube` as input and produces another `flimview` class called `FlimFit` which contains the fitting information for every pixel,

including all parameters from the model, their errors and χ^2 values. It is a collection of 2D arrays for the results of the fitting.

To generate a `FlimFit` object we can use the following:

```
# Define the boundaries for parameters
bounds_lower=[0.0, 0.0, 0., 0.]
bounds_upper=[1, 1., 5., 1.]
# Fit the whole cube to create a FlimFit object
Ffit = flim.fitCube(
    FCbin,
    mymodel,
    pfit,
    bounds=(bounds_lower, bounds_upper),
    norm=True,
    threshold=0.02)
```

Visualization. The package also includes means to visualize the `FlimCube` and `FlimFit` objects using intensity or peak values. Additionally, one can visualize the results from the fitting procedure using a function called `plotFit`, which takes a `FlimCube` and a `FlimFit` as input, along with a pixel coordinate, to produce a figure as the one shown in [Figure 4](#), showing the fit results for a given pixel, the parameters, the residuals, and the residuals distribution which is a useful way to explore the results. These can be combined in an interactive widget showing, for example, the same figure for a given point selected interactively. It is also possible to combine multiple visualizations to generate a sequence of plots into an animation of the datacube. The [package repository](#) includes examples on how this can be done to generate a short animation. Other visualization functions are also included and documented in the package.

Storage. `FlimCube` and `FlimFit` can be easily stored and easily retrieved by using a hierarchical and compression data store format for array-like data called HDF5 ([The HDF Group, 2000–2020](#)). Using this data format, we can store multiple data in different ‘folders’, including parameters, different models, masks, raw, binned and fitted data in one file for easy retrieval, as it only loads what is asked for, saving memory consumption if necessary. In the case of multiple files, we can serve them on-demand using a web server and only load the data as necessary, including over the web, which is an HDF5 feature. The methods `saveCube`, `saveFit`, `loadCube`, `loadFit`, `viewH5` are implemented in the `io_utils` modules inside `flimview`. For example, after running `fitCube` and saving the example data in a file, one can easily see its internal structure with:

```
import flimview.io_utils as io
io.viewH5(h5file)
```

Figure 4. Visualization example showing the fit for an individual pixel, the residuals and the location on the image for both, `.sdt` (a) and `.ptu` (b) files.

which produces the following output, showing the file content of the processed data:

```
File: test_ptu .h5
----> example_ptu: /example_ptu
-----> v0: /example_ptu /v0
-----> binned: /example_ptu /v0/ binned
-----> data: (256, 256, 107)
-----> mask: (256, 256)
-----> fit: /example_ptu /v0/ fit
-----> a1: (256, 256)
-----> a1_err: (256, 256)
-----> chi2: (256, 256)
-----> level: (256, 256)
-----> level_err: (256, 256)
-----> mask: (256, 256)
-----> residuals: (256, 256)
-----> tau1: (256, 256)
-----> tau1_err: (256, 256)
-----> tau2: (256, 256)
-----> tau2_err: (256, 256)
-----> raw: /example_ptu /v0/ raw
-----> data: (256, 256, 107)
-----> v1: /example_ptu /v1
-----> raw: /example_ptu /v1/ raw
-----> data: (256, 256, 107)
-----> v2: /example_ptu /v2
-----> raw: /example_ptu /v2/ raw
-----> data: (256, 256, 107)
-----> v3: /example_ptu /v3
-----> raw: /example_ptu /v3/ raw
-----> data: (256, 256, 107)
```

More examples on how to save and retrieve FLIM data using this format are included in the example notebooks at the [Github](#) repository.

Use cases

To highlight the main features of `flimview`, we have used two of the most common FLIM data formats, namely, the Becker and Hickl `.sdt`, and PicoQuant `.ptu`. We have included the reading routines for both data types but others can be easily extended within the module. All the figures in this paper were made using the two example files included in the package which we describe below. In particular [Figure 4](#) shows both files presented in the same visualization after being processed into a `FlimCube`. Both example files are included with the package and are briefly described below.

SDT File

The example file `epidermis.sdt` contains 2-photon excited FLIM data from the skin (epidermis) on the upper forearm of a healthy human volunteer captured with commercial optical medical imaging system (MPT-lex CARS, JenLab GmbH, Germany) ([Weinigel et al., 2015](#)). The image was captured using the procedure described in ([Alex et al., 2018](#)). For generating optical images, the excitation wavelength of the femtosecond laser was set to 760 nm and the incident in situ laser power was set to 30 mW. The light was focused through a 40x 1.35 NA objective. Autofluorescence signals within the spectral range of 405 nm–600 nm were detected. All these parameters are extracted from the file and added to the header inside the `FlimCube`.

PTU File

The example file `macrophages.ptu` contains the 2-photon excited FLIM data from J774A.1 mouse macrophages grown in Dulbecco's Modified Eagle Medium + 10% FBA + 1% antibiotic, under 5% CO₂. The image was acquired with the laser set to 750 nm and the incident in situ laser power was set to 25 mW.

Summary

In this paper we have presented `flimview`, a novel Python 3 package to manipulate, visualize, and analyze FLIM images and measurements from specific data formats `.sdt` and `.ptu`. We included snippets on how different

kernels and different masking strategies can be applied after reading the raw data into a `FlimCube` object. We also show how the procedure of fitting the whole cube is carried out and how these results can be visualized effectively. Most of the functions provided in this package can be customized for further research or to fit specific needs, including user defined binning kernels, mask functions or exponential decay fitting models. We showed how, using a hierarchical data format, the results from multiple files, multiple formats and multiple views can be stored in a single optimized data format called HDF5 which allows much better data management and data access to the data sets, even in the case of multiple files.

Data availability

We have included two data examples along with `Jupyter` notebooks to explore and get started with the module and its functions. This data examples can be found in the following repository: <https://github.com/Biophotonics-COMI/flimview>. Individual files can be loaded locally using this snippet. Detailed instructions and description of the data included can be found in the package website:

```
from flimview import datasets
# Get SDT example file
datasets.fetch_sdt()
# Get PTU file
datatsets.fetch_ptu()
```

Software availability

1. Package and installation instructions can be found here: <https://github.com/Biophotonics-COMI/flimview>
2. Source code can be found here: <https://github.com/Biophotonics-COMI/flimview>
3. Archived source code at time of publication: <http://doi.org/10.5281/zenodo.3825694> (Carrasco Kind, 2020).
4. License: **NCSA**

References

- Alex A, Frey S, Angelene H, *et al.*: **In situ** biodistribution and residency of a topical anti-inflammatory using fluorescence lifetime imaging microscopy. *Br J Dermatol*. 2018; 179(6): 1342–1350.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Ballesteros GC, Proux R, Bonato C, *et al.*: **readPTU: a python library to analyse time tagged time resolved data**. *Journal of Instrumentation*. 2019; 14(06): T06011–T06011.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Becker W, Bergmann A, Hink MA, *et al.*: **Fluorescence lifetime imaging by time-correlated single-photon counting**. *Microsc Res Tech*. 2004; 63(1): 58–66.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Bergmann A: **Spimage: data analysis software for fluorescence lifetime imaging microscopy**. *Becker & Hickl GmbH*. 2003.
[Reference Source](#)
- Borhani N, Bower AJ, Boppart SA, *et al.*: **Digital staining through the application of deep neural networks to multi-modal multi-photon microscopy**. *Biomed Opt Express*. 2019; 10(3): 1339–1350.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Bower AJ, Li J, Chaney EJ, *et al.*: **High-speed imaging of transient metabolic dynamics using two-photon fluorescence lifetime imaging microscopy**. *Optica*. 2018; 5(10): 1290–1296.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Bower AJ, Marjanovic M, Zhao Y, *et al.*: **Label-free in vivo cellular-level detection and imaging of apoptosis**. *J Biophotonics*. 2017; 10(1): 143–150.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Bower AJ, Sorrells JE, Li J, *et al.*: **Tracking metabolic dynamics of apoptosis with high-speed two-photon fluorescence lifetime imaging microscopy**. *Biomed Opt Express*. 2019; 10(12): 6408–6421.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Carrasco Kind M: **Biophotonics-COMI/flimview: flimview 1.2.0 (Version v1.2.0)**. *Zenodo*. 2020.
<http://www.doi.org/10.5281/zenodo.3825694>
- Cao R, Wallrabe H, Siller K, *et al.*: **Optimization of FLIM imaging, fitting and analysis for auto-fluorescent NAD(p)H and FAD in cells and tissues**. *Methods Appl Fluoresc*. 2020; 8(2): 024001.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Flimlib: a package for exponential curve fitting of fluorescence lifetime image data**. Accessed: 2020-02-26.
[Reference Source](#)
- Hirvonen LM, Suhling K: **Wide-field TCSPC: methods and applications**. *Meas Sci Technol*. 2016; 28(1): 012003.
[Publisher Full Text](#)
- McGinty J, Talbot CB, Owen D, *et al.*: **Fluorescence lifetime imaging: Microscopy, endoscopy, and tomography**. In: *Handbook of Biomedical Optics*. CRC Press, 2016; 609–636.
[Reference Source](#)
- Ranawat H, Pal S, Mazumder N: **Recent trends in two-photon auto-fluorescence lifetime imaging (2p- flim) and its biomedical applications**. *Biomed Eng Lett*. 2019; 1–18.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- The HDF Group: **Hierarchical data format version 5, 2000-2020**.
[Reference Source](#)
- Warren SC, Margineanu A, Alibhai D, *et al.*: **Rapid global fitting of large fluorescence lifetime imaging microscopy datasets**. *PLoS One*. 2013; 8(8): e70687.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Weinigel M, Georg Breunig H, Uchugonova A, *et al.*: **Multipurpose nonlinear optical imaging system for in vivo and ex vivo multimodal histology**. *J Med Imaging*. 2015; 2(1): 016003.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)

Open Peer Review

Current Peer Review Status:

Version 1

Reviewer Report 01 September 2020

<https://doi.org/10.5256/f1000research.26482.r68967>

© 2020 Mazumder N. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Nirmal Mazumder

Department of Biophysics, Manipal School of Life Sciences, Manipal Academy of Higher Education (MAHE), Manipal, Karnataka, India

The manuscript on flimview is well written and rationale of developing a new software is needed to analysis FLIM data acquired by commercial software or hardware.

The manuscript can be accepted for indexing after minor revision.

Line 4, in introduction, It should be "TCSPC".

In this manuscript, definition of "fluorescence lifetime" is missing.

Author could describe in brief about the instrumentation. Is flimview applicable for single photon or two photon fluorescence measurement?

A step by step flowchart about the flimview could help to implement.

In figure 2, only center of the image is used for analysis after masking. However, in some cases cells are distributed in full field of view. In such case, how does this kind of masking work?

Author should mention in figure 2, what kind of image it is, what does the color mean, what type of cell is imaged.

In fitting, why two lifetimes are used for fitting? If there is only one flurophore then single exponential fitting may work. In case of NADH, it exists in free and bound forms and hence two lifetimes. Author should comment on it.

In figure 4, it will be more acceptable if author can compare the results of flimview with Becker and Hickl or PicoQuant software analysis. Residues should be close to 1. Is it due to experimental or software error?

Is the rationale for developing the new software tool clearly explained?

Yes

Is the description of the software tool technically sound?

Partly

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Partly

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Biophotonics, Biophysics

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Reviewer Report 02 July 2020

<https://doi.org/10.5256/f1000research.26482.r65063>

© 2020 Michalet X. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Xavier Michalet

Department of Chemistry and Biochemistry, University of California, Los Angeles (UCLA), Los Angeles, CA, USA

Reviewer's Report on the manuscript "flimview..." by Kind et al.

This paper describes a python package to load and visualize FLIM datasets. The source code has been deposited on Github, together with two example data files used to illustrate some of the software features.

The manuscript is well written and describes the software features with a fair amount of detail. The samples of code I have reviewed appear to be also reasonably well documented. Based on these basic criteria, this work should be valuable as a basis for future developments by researchers interested in developing their own tools for FLIM analysis.

I would however encourage the authors to address the following remarks:

1. Presentation:

- For whatever reasons, the figures do not include color scales, which are present in the Jupyter notebooks used to generate them. This can easily be fixed and would be useful.
- The datasets shown correspond to microscopy images of biological samples. It is customary (and very useful) to show such data with a scale bar. Consider making the extra effort to generate one (maybe optionally) as part of the code, to encourage scientific rigor.
- Some of the features of the software are incompletely described. For instance, the package opens .sdt and .ptu files *containing photon timestamps*, but does not dwell on the choice of timestamp binning used internally. This of course has an effect on RAM resources but also potentially on decay artifacts. As a side comment, the package, as is, does not support pre-binned datasets, as far as I can tell (see below). Some .sdt files are pre-binned only and would not be handled by the current software version.
- I am not sure what the authors mean by “after compensating for the instrument response function” (p 3)? Clearly the code does fit the decays without IRF convolution (which of course requires the user to provide an IRF file as well).
- While I understand why binning the original dataset could be interesting to increase SNR and decrease data footprint, I am unclear about the role of convolution with Gaussian or other kernels. What would this correspond to physically? I would understand deconvolution of the image with such kernels (although that would not address the question of how the photon time stamps should be redistributed in the deconvolved image), but the purpose of these kernels, besides image beautification, is mysterious and not an encouraged practice for rigorous analysis.

2. Content:

- While this software is understandably only offering basic features (opening datasets – 2 kinds only supported, pre-processing, displaying an intensity image and fitting), it seems to be missing what I would consider an expected functionality for a software bearing the *flimview* name, namely visualization of fluorescence lifetime information. A barebone commercial software such as SPCImage from Becker & Hickl, at least does offer a mean lifetime representation, which would not seem to be difficult to implement. Since we are dealing with a software intended for research, I would have naively expected that options to represent the bi-exponential fitted amplitudes and lifetimes as color-coded maps would have been offered.
- The software allows saving the data in the open and multiplatform HDF5 format. While this is commendable, it is not quite clear what principles are governing the structure of the data in this file format, or for that matter, what the file format actually looks like from a bird’s eye view. Our group has made an attempt to offer a similar type of HDF5-based file format for a slightly different purpose (*photon-HDF5*), which could be taken as an example (if maybe extreme in its level of detail) of what could be provided to potential users¹. More importantly, it is not quite clear how this file format will and could evolve, whether it is intended to become a standard, while it has been chosen preferentially to another, etc. A minimum discussion of these

considerations within the manuscript, and a more extended presentation online (say on the Github website) would be extremely useful.

- It would be useful to hear from the authors whether and how they intend to encourage contribution and ensure evolution of the software. Of particular importance would be evidence that there is some kind of quality control (test modules – e.g. such as those described in a software cited in the file loading code, [FRETburst²](#)- written using [pytest](#)).
- Also related to the lack of bird's eye view description is the absence of discussion of the principles involved in defining the FlimCube class. How can it be extended to support more than one spectral channel (or any other spectroscopic variable)? An additional spatial dimension? An additional temporal dimension?

References

1. Ingargiola A, Laurence T, Boutelle R, Weiss S, et al.: Photon-HDF5: An Open File Format for Timestamp-Based Single-Molecule Fluorescence Experiments. *Biophys J.* 2016; **110** (1): 26-33 [PubMed Abstract](#) | [Publisher Full Text](#)
2. Ingargiola A, Lerner E, Chung S, Weiss S, et al.: FRETbursts: An Open Source Toolkit for Analysis of Freely-Diffusing Single-Molecule FRET. *PLoS One.* 2016; **11** (8): e0160716 [PubMed Abstract](#) | [Publisher Full Text](#)

Is the rationale for developing the new software tool clearly explained?

Yes

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: single-molecule FRET, wide-field FLIM, single-photon detectors

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

The benefits of publishing with F1000Research:

- Your article is published within days, with no editorial bias
- You can publish traditional articles, null/negative results, case reports, data notes and more
- The peer review process is transparent and collaborative
- Your article is indexed in PubMed after passing peer review
- Dedicated customer support at every stage

For pre-submission enquiries, contact research@f1000.com

F1000Research