AlliGator: Open Source Fluorescence Lifetime Imaging Analysis in G

X. Michalet[a]

[a]*Department of Chemistry & Biochemistry*
University of California at Los Angeles
607 Charles E. Young Drive E
Los Angeles, CA 90095
michalet@ucla.edu

Abstract

Fluorescence Lifetime Imaging (FLI) is a technique recording the temporal decay of fluorescence emission at every pixel of an image. Analyzing the information embedded in FLI dataset requires either fitting the decay to a predefined model using nonlinear least-square fit or maximum likelihood estimation, or projecting the decay on an orthogonal basis of periodic functions as in phasor analysis. AlliGator is an open source software implementing these approaches in a user-friendly graphical user interface (GUI) offering numerous unique features not found in other existing packages. It leverages the unique ability to design user-friendly GUI of the LabVIEW graphical programming language (G), and supports user-developed plugins written in python.

Keywords: fluorescence lifetime imaging, NLSF, phasor analysis, LabVIEW, python

Metadata

1. Motivation and significance

Fluorescence lifetime imaging (FLI) techniques have gained in popularity in the biomedical field thanks to technological advances in laser sources and detectors, both in its microscopic version (FLI microscopy or FLIM) [1] and its mesoscopic or macroscopic version (mFLI or MFLI) [2, 3, 4, 5]. However, FLI data analysis remains challenging as different analytical approaches are available, such as nonlinear least-square fit (NLSF) [6], maximum likelihood estimation (MLE) [7], phasor analysis [8] or deep learning [9], each requiring expertise to implement, and once data is processed, to interpret.

Nr.	Code metadata description	Metadata
C1	Current code version	1.0.2
C2	Permanent link to code/repository used for this code version	`https://github.com/smxplorer/alligator`
C3	Permanent link to Reproducible Capsule	`https://doi.org/10.5281/zenodo.15468169`
C4	Legal Code License	BSD
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	LabVIEW 2021, Python 3.9-3.10
C7	Compilation requirements, operating environments & dependencies	LabVIEW 2021 SP1 64-bit, Vision Development Module 2021, OpenG Toolkit, etc.
C8	If available Link to developer documentation/manual	`https://github.com/smXplorer/AlliGator/tree/main/src-docs`
C9	support email	michalet@ucla.edu

Table 1: Code metadata (mandatory)

Nr.	(Executable) software metadata description	Please fill in this column
S1	Current software version	1.0.2
S2	Permanent link to executables of this version	`https://github.com/smXplorer/AlliGator/tree/main/installer`
S3	Permanent link to Reproducible Capsule	`https://doi.org/10.5281/zenodo.15468169`
S4	Legal Software License	BSD
S5	Computing platforms/Operating Systems	Microsoft Windows
S6	Installation requirements & dependencies	LabVIEW 2021 SP1 64-bit Runtime Executable, Vision Development Module 2021 Runtime Executable
S7	link to user manual	`https://alligator-distribution.readthedocs.io`
S8	Support email for questions	michalet@ucla.edu

Table 2: Software metadata (optional)

AlliGator was developed with the aim of providing maximum flexibility to the user in exploring the effect of changing parameters on the results of different analysis approaches, while keeping the interface purely graphical and interactive. It has been used in scientific publications covering techniques as diverse as wide-field m/MFLI using an intensified CCD camera [10, 11], a linear SPAD array [12] and a SPAD camera [13], or FLIM using time-resolved confocal microscopy [14] or a SPAD camera [15], and is continuously being improved with additional features.
The software takes advantage of the simplicity of developing feature-rich graphical user interface (GUI) in the graphical programming environment LabVIEW (a trademark of Emerson/National Instruments) and its image handling library (sold under the trade name `Vision Development Module` or `VDM`). The software comprises a main window, which provides different visualizations of the dataset under analysis (left side of Fig. 1) as well as as of different analysis outputs in linked panels (right side of Fig. 1). Accessory windows allow the user to set analysis parameters, explore the dataset at the single-pixel or single-region of interest (ROI) level, perform line profile analysis or manipulate and fit histograms of multiple analysis outputs. The user interacts with the intuitive GUI using buttons, window menus and graphical objects' contextual menus, as well as via mouse-based actions on images, graphs, slides and most other graphical UI objects.
AlliGator distinguishes itself from a number of similar projects (a fairly comprehensive list of which can be found on the documentation page of the PhasorPy project [16]) by supporting datasets requiring per-pixel instrument response function or phasor calibration (as is required with SPAD cameras), as well as by numerous ways of analyzing data from multiple ROIs, both in terms of fitting methods and the phasor approach. While its core is developed using the graphical programming language LabVIEW, a simple API to support python plugins is included, which does not require any access to the LabVIEW source code or compiler. An installer for a standalone version of the software is released with each new version, that only requires a license for the VDM runtime.

2. Software description

2.1. Software architecture

AlliGator is built around a simple producer-consumer architecture. The main producer is the user-event loop hosted by the main window (see Fig. 1 for the UI and Fig. 2 for an overview of the software architecture and Fig. A.6 for a complete hierarchy of the different libraries), which registers all user

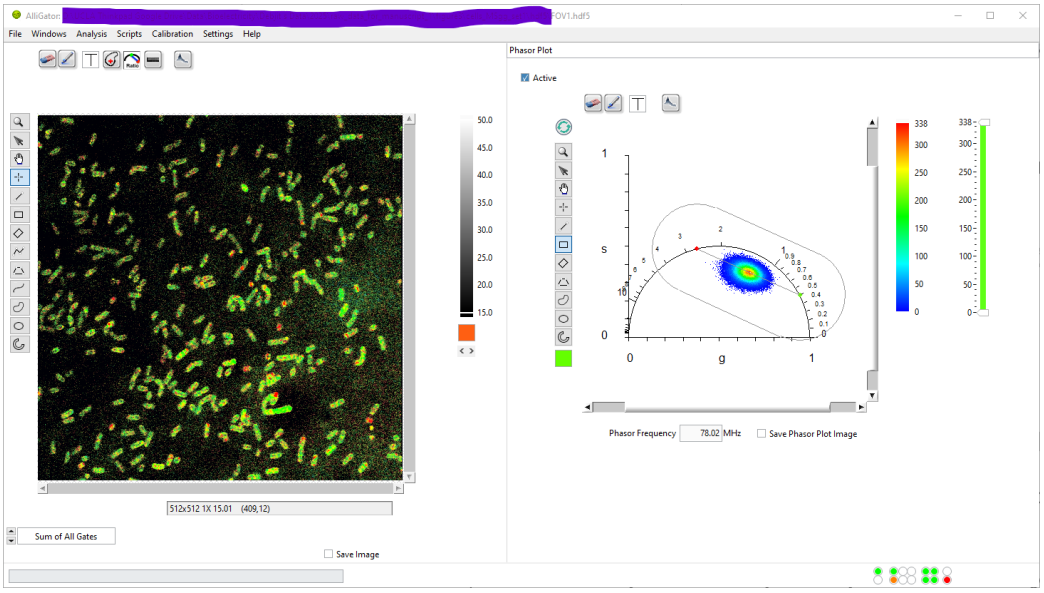

Figure 1: AlliGator's main window during the final stage of an example of FLI analysis described in the text. The left side of the window shows a dataset after overlay of the color-coded amplitude-averaged lifetime. The right side of the window show the `Phasor Plot` panel in which the pixel-wise phasor plot is represented. Two references (green and red dots) used for linear decomposition of each phasor are also represented, together with the segment connecting them and the region within which linear decomposition analysis is confined.

interface interactions and translates them into action requests added to the main AlliGator action queue (green Q icon in Fig. 2).
The consumer is a finite state machine loop (shown on the right of Fig. 2) running in a separate routine (or *Virtual Instrument* as an individual piece of LabVIEW graphical code is called, or *VI* for short), which dequeues action requests and processes them in a first-in first-out order.
Ancillary windows (`Histogram`, `Settings`, `Image Profile`, `Local Decay Graph`, etc.) launched from the main AlliGator window by user actions such as selection of window menu items or right-click on contextual menu items, behave in the same manner, sending action requests via the main queue, and receiving UI update data via dynamic events.
Internal data, including the current dataset and results of different analysis steps, are stored in a single structure passed along as a reference to sub-VIs involved in data processing, avoiding unnecessary data copies. User-defined settings can be interactively changed via a dedicated multi-panel `Settings` window, these settings being accessible (and modifiable) anywhere in the code.
Thanks to the built-in parallelism of LabVIEW code, a number of computa-

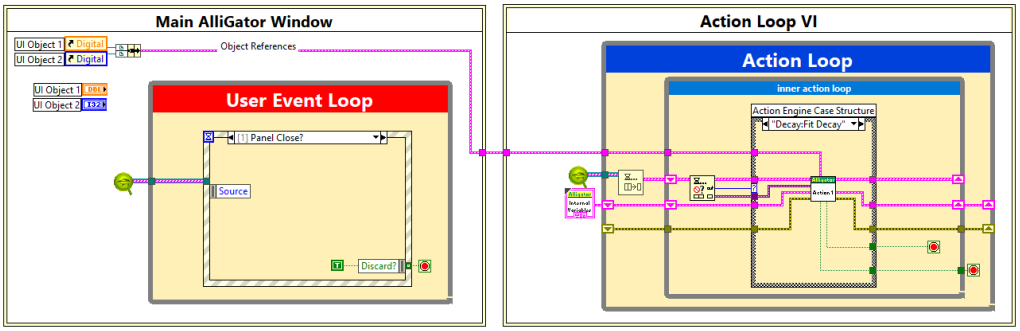

Figure 2: Simplified architecture of the AlliGator software. The `Main AlliGator Window` virtual instrument (VI), whose simplified code is shown to the left, contains a user-event loop which registers user interactions with various UI objects and passes corresponding action requests via a queue (green Q icon) to the `Action Loop VI` (shown on the right) running in parallel. Ancillary windows opened by the `Main AlliGator Window` (not shown) also send action requests to the `Action Loop VI`. The action loop handles *scripts* (arrays of action requests) in a first-in first-out order, and can insert new action requests within a script. Actions are processed by a hierarchy of sub-VIs organized into specialized libraries (e.g. file loading, phasor analysis, NLSF, etc.). Update requests to ancillary windows are handled via custom user-events. Note that in the actual code, the two pieces of code shown in the two boxes are disconnected (run in parallel), the references to the main window's objects being passed dynamically at launch time to the `Action Loop VI`.

tional tasks such as NLSF calculations can take advantage of the computer's multicore architecture and multithreading capabilities, speeding up analysis of large datasets.

2.2. Software functionalities

AlliGator supports datasets files from major FLI data acquisition hardware, including confocal microscopes or intensified CCD (ICCD) and wide-field SPAD cameras. These datasets can consist of either streams of photons with a position, macrotime and microtime information for each photon, or a series of images, each image corresponding to a finite temporal interval after the laser excitation, from which the fluorescence decay at each pixel can be reconstructed. It also allows saving datasets into a highly efficient and open source file format based on the Hierarchical Data Format (HDF5) (see the corresponding *AlliGator HDF5 File Format* online manual page for details [17]). Analysis results can be saved as text files (e.g. comma-separated-values or CSV) for further processing by third party software, or reloaded for further processing in AlliGator. Information pertaining to ROIs, phasor calibration, phasor references or detailed phasor or NSLF analysis outputs are saved in easy to read extended markup language (XML), Javascript Object Nota-

tion (JSON) or HDF5 file formats, which can be reloaded for reproducibility purposes.
Datasets can be averaged, binned, background corrected or pile-up corrected when applicable, and single pixel or region of interest (ROI) decays can be processed in a variety of optional ways easily configurable via a dedicated `Settings` window.
AlliGator supports multiple user-defined ROI shapes as well as algorithmically defined ones (grids, ROIs with pixel intensities over a threshold, image mask definitions, etc.), which can be augmented with user-defined python plugins if necessary. ROIs can be merged or on the contrary, decomposed into individual pixel ROIs for additional insights.
Single or multiple ROIs, as well as all pixels of an image, with or without additional constraints on intensity or peak value, can be analyzed using single- or double-exponential fits using weighted or unweighted NLSF or MLE, as well as by phasor analysis. Maps of fit parameters (lifetimes, amplitudes, χ^2, R^2, etc.) or phasor-derived quantities (fraction of a 2-component mixture, average lifetime, etc.) can be overlaid as color-coded maps on the source image, exported as images or ASCII matrix files, or further processed to extract statistical histograms or scatter-plots of one parameter versus another, among other analysis workflows described in the online manual [17]. These operations are accessible via right-click menus in the respective graphs representing the results of different analysis steps. Histograms of parameters can be fitted with various (multi)peak models or exported as text files for external processing.
Similarly, phasor analysis can be performed at the single or multiple ROIs, or full image level. ROIs can be defined within the phasor plot as well, in order to highlight the corresponding regions of the source image, or to limit further data analysis to a subset of the computed phasors. AlliGator offers different ways to define the so-called phasor calibration at the core of any phasor analysis, and allows saving calibration to file for further reuse. It also provides a list of phasor processing functions.
Most importantly, every single action and analysis output is automatically exported into a rich text format (RTF) notebook, in which the user can also copy and paste any image from AlliGator's many windows, or from external sources, as well as type any text needed to further document an analysis workflow.
Finally, AlliGator's functionalities can be extended with Python plugins, which are seamlessly added to AlliGator main menu or local objects' menus as described in the *Python Plugins* menu pages of the online manual [17].

2.3. Sample code snippet analysis

Fig. 3 shows a graphical code snippet (or diagram) corresponding to one example of data analysis in AlliGator. This particular analysis determines the intersection of the principal axis of inertia of the phasors with the universal circle (UC), that is, the locus of single-exponential decays in the phasor plot [8]. The corresponding two phasors (corresponding to two single-exponential decays and referred to as *phasor references*) can then be used for linear decomposition of any phasor into a weighted sum of these references.

This VI takes as inputs the internal variable's data value reference (`AlliGator IV DVR`), the `phasor frequency` as well as a boolean flag (`Major Axis?`) specifying which principal axis to use for the analysis. It then retrieves the necessary internal data (2D array of `Valid Pixels?` flags, 2D array `CSG Phasors` of single-precision complex phasor values), and user-defined settings defining whether to limit the analysis to one or more of the ROIs defined in the phasor plot image, and computes the major and minor axis before determining the intersection of the selected axis with the UC. The resulting intersections (`Phasor 1` and `Phasor 2`) are then output together with an optional string (`Message`) describing the phasor ROIs used in the analysis (empty if none were used).

Subsequent VIs (not shown) called after this code snippet display these phasor references in the phasor plot image and graph, and save them internally.

This code snippet illustrates the clean and visual simplicity of properly laid out graphical code, each square icon with a banner in the diagram representing a sub-VI (sub-routine), whose graphical code can be accessed by double-clicking it in the LabVIEW development environment, this process repeating itself until only native functions are involved in the sub-VI's code.

The full list of VIs comprising AlliGator and a brief description of their function can be found in the `src-docs` folder of the source code repository [18].

3. Illustrative example

Here, we briefly provide an overview of the type of analysis used in a recent article studying the membrane potential of bacteria using voltage sensitive fluorophores whose lifetime depends on the local electric field [14]. The bi-exponential nature of the fluorescence decays observed in this study required a two-component phasor analysis involving:

- loading the dataset
- defining ROIs corresponding to individual bacteria

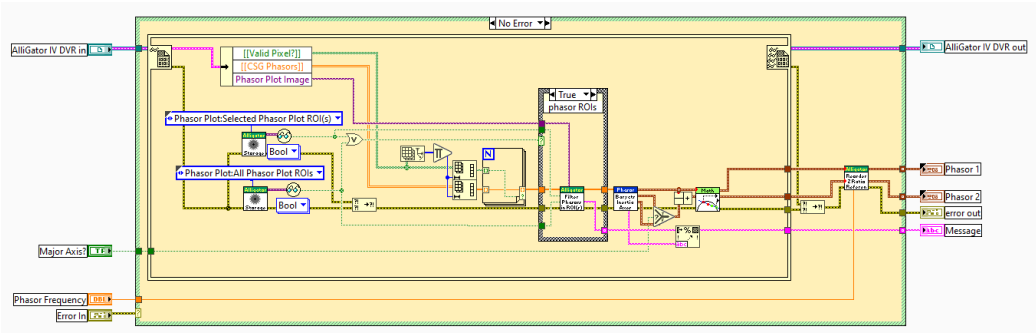

Figure 3: Example of AlliGator code: `AlliGator Use Phasor Plot UC-Principal Axis Intersections as References.vi`. This snippet computes the minor or major axis of the phasor plot and determines its intersection with the universal circle (UC), the locus of single-exponential decays in the phasor plot, and uses the two corresponding phasors as references for subsequent mixture analyses.

- defining the calibration phasor
- computing the calibrated phasor of each ROI
- determining the phasor references to be used for mixture decomposition
- computing the average lifetime of individual ROIs based on these references

The first step of loading the dataset (an example of which can be found as part of the FigShare data repository [19] associated with Fig. 3 of ref. [14]) involves a mere drag and drop of the corresponding file (cells_MSgg_set1 \hdf5 \FOV1.hdf5) into the main AlliGator window, resulting in the display of the intensity image (sum of all photons detected at each pixel, irrespective of their arrival time, see Fig. 4a).
The second step consists in defining ROIs identifying each bacterial cell. In this work, we used the intensity image shown in Fig. 4a as the input to the StarDist plugin [20] of Fiji [21] to automatically identify the bacteria, and exported the corresponding label image as a TIFF file (cells_MSgg_set1 \mask \mask_FOV1.tif). This TIFF file is then loaded into AlliGator as a *mask image*, with the different ROIs automatically highlighted by different colors in Fig. 4b.
After having specified the correct phasor frequency f ($f = 1/T = 78.02$ MHz, where T is the laser period), and computed the phasor of the instrument's instrument response function (IRF_decay.txt), the IRF phasor was set as the calibration phasor (phasor associated with lifetime $\tau = 0$ using the AlliGator *Phasor Graph*'s right-click menu `Use Current Phasor as Calibration`

item. This step is typically done once per analysis session, and can be bypassed altogether by saving the calibration for future reuse.
Once the phasor calibration is defined and activated by toggling the *Calibration Type* in the Phasor Graph panel to `Single Phasor`, the `Non-Interactive (Fast)` script in the `Analysis: FLI Dataset: Multiple ROIs Analysis: All ROIs Phasor Analysis` menu computes the phasors of each ROI (whose pixel signal are summed up in each ROI) and plots them as a single scatterplot in the Phasor Graph panel (Fig. 4c).
This allows defining "natural" single-exponential components describing the set of similar cells using an AlliGator-specific tool, namely the calculation of the major axis of inertia of the phasor scatterplot (see section 2.3 and Fig. 3), and determining its intersections with the universal circle (indicated as green and red dots in Fig. 4c).
Finally, the last step consists in using these references to compute the amplitude-averaged lifetime $\langle\tau\rangle_a$ of each ROI [14] and histogram their values to obtain the mean and standard deviation of the observed lifetime in the bacterial population (Fig. 4d).
A similar analysis obtained by NLSF analysis of the same ROIs can be performed using the corresponding `Non-Interactive (Fast)` script in the `Analysis: FLI Dataset: Multiple ROIs Analysis: All ROIs NLSF Analysis` menu.

4. Impact

AlliGator is designed as a graphical UI tool in order to save the user the need to write or edit scripts, a popular approach used for instance in Jupyter Notebooks, but that can lead to typographic or syntax mistakes resulting in errors, which are not always easy to diagnose. It is designed to analyze data in terms of single- or double-exponential decays, either by NLSF, MLE or phasor analysis.
It can be used for instance to analyze mixtures of fluorophores characterized by different lifetimes, and in particular, mixture of fluorescence resonance energy transfer (FRET) donor-acceptor pairs with donor only species, as demonstrated in a series of recent works [10, 15, 22, 13, 23, 14]. It allows performing pixel-wise, ROI-wise, multi-ROIs or full dataset analysis seamlessly, with the simple selection of a menu item.
While it is fairly complete at this stage, it is in no way exhaustive. For instance, to compensate for its current limitation in terms of publication-quality graph outputs, all results can be exported in text format from all graphs for further editing or processing with 3rd party software. Likewise, if a feature is lacking, users with python programming expertise can easily add

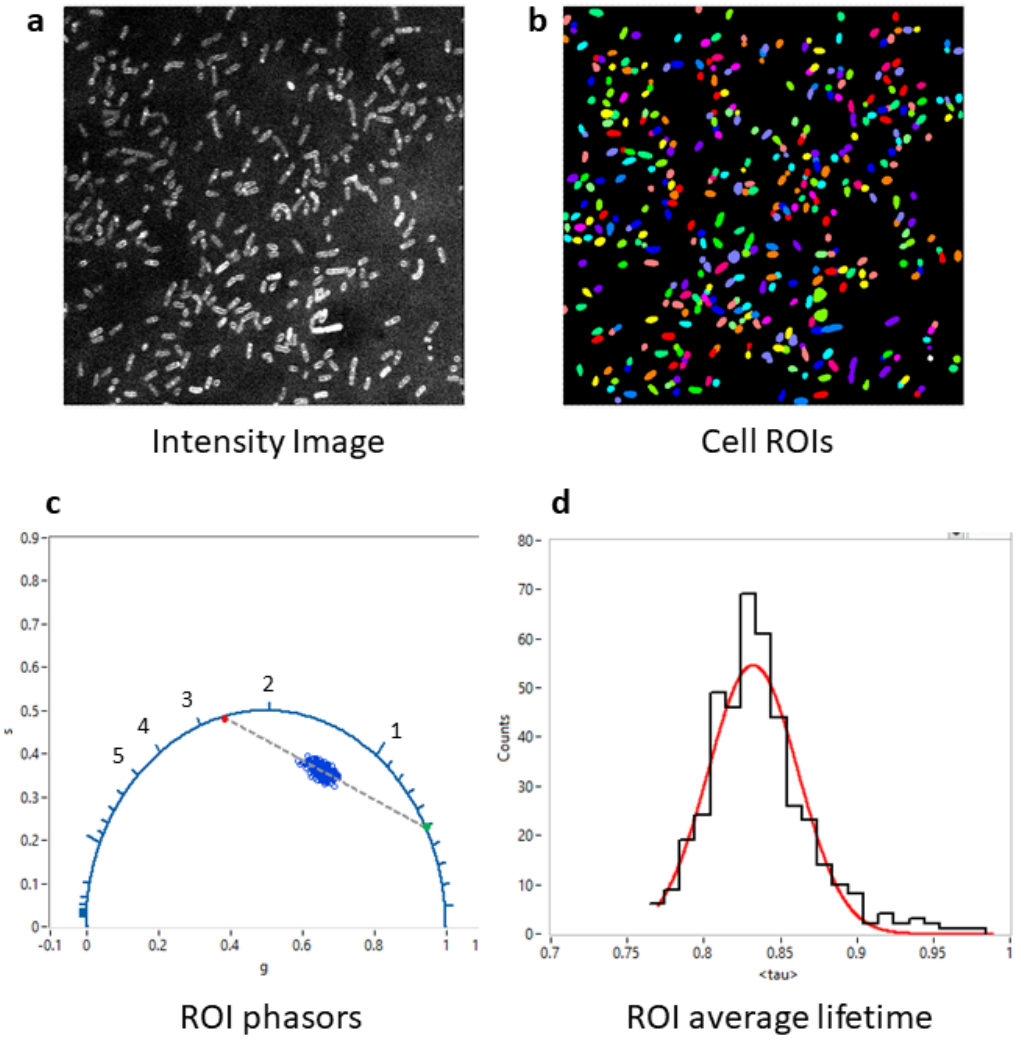

Figure 4: Example of phasor analysis using AlliGator. After loading the dataset (a) and defining ROIs (b), the phasor of each ROI is calculated after calibration with the appropriate instrument response function (c). A function of AlliGator's `Phasor Graph` allows computing single-exponential *references* as the intersection of the universal circle (blue circular arc) and the major axis of the phasor scatter plot (green and red dots connected by a dashed line. These references can then be used to decompose each phasor into a linear combination of both of them, from which the amplitude-averaged lifetime $\langle\tau\rangle$ can be computed for each phasor. The histogram in (d) is the result of a two-step analysis: first, the creation of a $\langle\tau\rangle(I)$ plot and next, the calculation of the histogram of $\langle\tau\rangle$'s and its fitting with a Gaussian distribution. All images and graphs are direct export from AlliGator to the clipboard. The raw data for each plot can also be exported as an ASCII file for further analysis by third party software.

functionalities in the form of plugins, which are then seamlessly added to the software main menu or to specific objects (graph, image) in the software, without any need for recompilation.
AlliGator has been developed over more than a decade to address specific research needs in topics as diverse as *in vivo* FRET studies in live mice [10, 13] or membrane potential measurements in bacteria [14], using techniques as different as raster-scanning confocal microscopy [14] or wide-field single-photon avalanche diode cameras [15, 13]. It has been used in many articles as the tool of reference to which to compare alternative methods (e.g. deep learning) for FLI data analysis.
The software goes beyond the usual features offered by FLI analysis software, each function being documented in an extensive online manual created using open source tools (Sphynx [24] and ReadTheDocs [25]) and under version control using Git. It is designed with reproducibility in mind, all intermediate results and analysis steps being logged in a rich text format notebook window, to which the user can add or remove whatever piece of information they need. Its ease of use and performance has made it the tool of choice for our laboratory and our collaborators, who are using it on a daily basis to analyze data [10, 15, 22, 13, 12, 11, 26, 27, 23, 28, 29, 30, 31, 32, 14, 33] and as a benchmark against other methods such as newly developed deep learning algorithms [23].
Up until this source code release, AlliGator has been freely available as a standalone executable installer, initially on a Google Site hosted by UCLA (now discontinued) and since early 2023 on GitHub [18]. While GitHub does not provide long term visitor counts, the repository is regularly visited as indicated by the Insights repository page. This publication describes the first open source release of AlliGator.

5. Conclusions

AlliGator offers a powerful platform for FLI data analysis by NLSF or MLE bi-exponential fits and the phasor approach. The release of its graphical source code provides the opportunity for users to check algorithmic implementation, as well as potentially suggest performance improvements or additional functionalities, either to facilitate implementation of python plugins, or by contributing new LabVIEW code to the software. To our knowledge, it is one of the rare large size scientific software projects using this unique programming language to be released as an open source project, and with an extensive online manual and documentation.

Acknowledgments

This work was supported in part by Human Frontier Science Program Grant RGP0061/2015, US National Institute of Health Grants R01 GM095904 & R01 CA250636, University of California CRCC Grant CRR-18-523872, US Department of Energy Grant DE-SC0020338 & DE-SC0023184 the European Research Council under Europe-an Union's Horizon 2020 research and innovation program under grant agreement No. 669941 and by the Partner University Fund, a program of the French American Culture Exchange.
It is my pleasure to acknowledge the critical feedback, bug reports and feature suggestions from all users of AlliGator over the years, including current and former members of the Weiss lab at UCLA (Rinat Ankri, Debjit Roy, Kiran Bharadwaj), the Intes lab at RPI (Sez-Jen Chen, Jason Smith, Vikas Pandey, Nanxue Yuan, Ismail Erbas, Saif Ragab) and the Barroso lab at AMC (Catherine Sherry). Their expert observations and patience while waiting for improvements or bug fixes is deeply appreciated and was instrumental in the continued improvements of the software.

Appendix A. AlliGator VI Hierarchy

Fig. A.5 shows a tree view of the relationship between virtual instruments (VI) libraries (large rectangles with icons) and individual VIs (small icons) comprising AlliGator. The same tree view with expanded libraries is shown in Fig. A.6. A full resolution and navigable tree view can be obtained using the built-in LabVIEW tool `View:VI Hierarchy` with the `AlliGator.vi` VI opened.
Libraries and VIs specific to AlliGator are described in the `Source Code Documentation` found in the `src-docs` folder of the GitHub repository [34]. Ancillary libraries' VIs can be found in corresponding folders of the distribution. Third party libraries are listed in the Readme.md file accompanying the distribution.

References

[1] J. A. Levitt, D. R. Matthews, S. M. Ameer-Beg, K. Suhling, Fluorescence lifetime and polarization-resolved imaging in cell biology, Current Opinion in Biotechnology 20 (1) (2009) 28–36. doi:10.1016/j.copbio.2009.01.004.

[2] V. Venugopal, J. Chen, F. Lesage, X. Intes, Full-field time-resolved fluorescence tomography of small animals, Optics Letters 35 (19) (2010) 3189–3191. doi:10.1364/OL.35.003189.

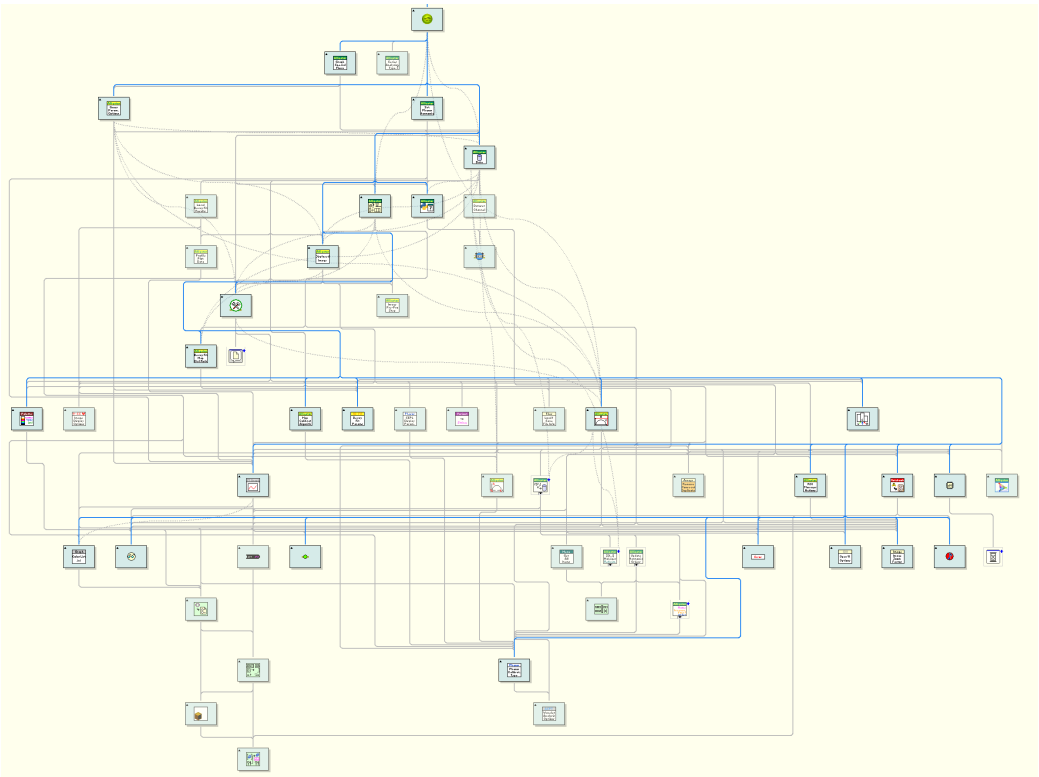

Figure A.5: Overview of the relationship between VI libraries comprising AlliGator.

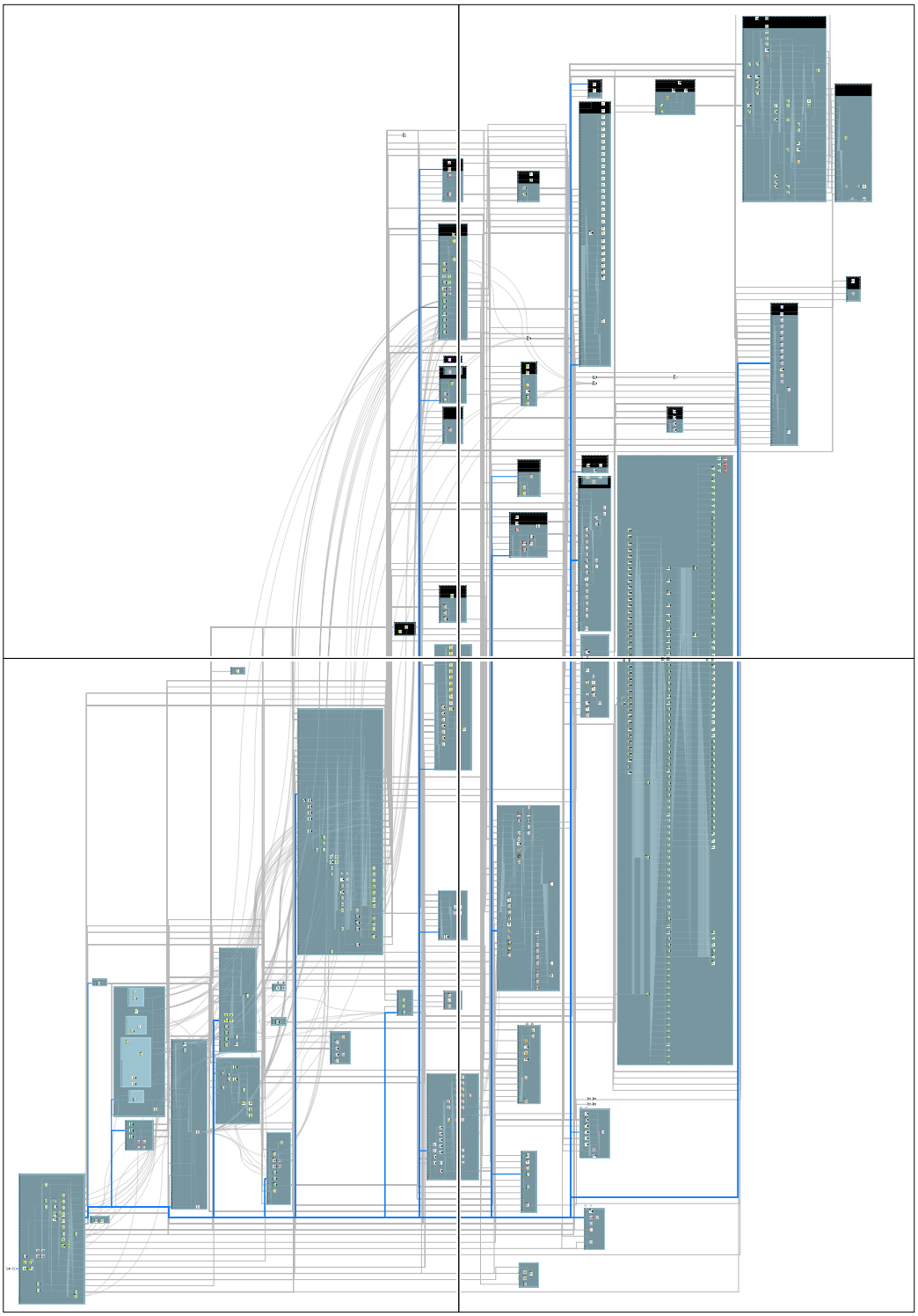

Figure A.6: Expanded overview of the relationship between the VIs comprising AlliGator.

[3] A. Alfonso-Garcia, J. Bec, B. Weyers, M. Marsden, X. Zhou, C. Li, L. Marcu, Mesoscopic fluorescence lifetime imaging: Fundamental principles, clinical applications and future directions, Journal of Biophotonics 14 (6) (2021). doi:10.1002/jbio.202000472.

[4] A. J. Bowman, C. Huang, M. J. Schnitzer, M. A. Kasevich, Wide-field fluorescence lifetime imaging of neuron spiking and subthreshold activity in vivo, Science 380 (6651) (2023) 1270–1275. doi:10.1126/science.adf9725.

[5] R. Pal, M. Krishnamoorthy, A. Matsui, H. Kang, S. Morita, H. Taniguchi, T. Kobayashi, A. Morita, H. S. Choi, D. G. Duda, A. T. N. Kumar, Fluorescence Lifetime Imaging Enables In Vivo Quantification of PD-L1 Expression and Intertumoral Heterogeneity, Cancer Research 85 (3) (2025) 618–632. doi:10.1158/0008-5472.Can-24-0880.

[6] H. P. Good, A. J. Kallir, U. P. Wild, Comparison of fluorescent lifetime fitting techniques, Journal of Physical Chemistry 88 (22) (1984) 5435–5441. doi:10.1021/j150666a066.

[7] P. Hall, B. Selinger, Better estimates of exponential decay parameters, Journal of Physical Chemistry 85 (20) (1981) 2941–2946. doi:10.1021/j150620a019.

[8] M. A. Digman, V. R. Caiolfa, M. Zamai, E. Gratton, The phasor approach to fluorescence lifetime imaging analysis, Biophysical Journal 94 (2) (2008) L14–L16. doi:10.1529/biophysj.107.120154.

[9] J. T. Smith, R. Yao, N. Sinsuebphon, A. Rudkouskaya, N. Un, J. Mazurkiewicz, M. Barroso, P. Yan, X. Intes, Fast fit-free analysis of fluorescence lifetime imaging via deep learning, Proceedings of the National Academy of Sciences 116 (48) (2019) 24019–24030. doi:10.1073/pnas.1912707116.

[10] S.-J. Chen, N. Sinsuebphon, A. Rudskouskaya, M. Barroso, X. Intes, X. Michalet, In vitro and in vivo phasor analysis of stoichiometry and pharmacokinetics using short lifetime near-infrared dyes and time-gated imaging, Journal of Biophotonics 12 (3) (2019) e201800185. doi:10.1101/212977.

[11] J. T. Smith, N. Sinsuebphon, A. Rudkouskaya, X. Michalet, X. Intes, M. Barroso, In vivo quantitative fret small animal imaging : Intensity versus lifetime-based fret, Biophysical Reports 3 (2) (2023) 100110. doi:10.1016/j.bpr.2023.100110.

[12] S. Gao, M. Li, J. T. Smith, X. Intes, Design and characterization of a time-domain optical tomography platform for mesoscopic lifetime imaging, Biomedical Optics Express 13 (9) (2022). doi:10.1364/boe.460216.

[13] J. T. Smith, A. Rudkouskaya, S. Gao, J. M. Gupta, A. Ulku, C. Bruschini, E. Charbon, S. Weiss, M. Barroso, X. Intes, X. Michalet, In vitro and in vivo NIR fluorescence lifetime imaging with a time-gated SPAD camera, Optica 9 (5) (2022) 532–544. doi:10.1364/OPTICA.454790.

[14] D. Roy, X. Michalet, E. W. Miller, K. Bharadwaj, S. Weiss, Towards measurements of absolute membrane potential in bacillus subtilis using fluorescence lifetime, Biophysical Reports 5 (1) (2025) 100196. doi:10.1016/j.bpr.2025.100196.

[15] A. Ulku, A. Ardelean, M. Antolovic, S. Weiss, E. Charbon, C. Bruschini, X. Michalet, Wide-field time-gated SPAD imager for phasor-based FLIM applications, Methods and Applications in Fluorescence 8 (2) (2020) 024002. doi:10.1088/2050-6120/ab6ed7.

[16] PhasorPy (2025).
URL https://www.phasorpy.org/docs/v0.5/phasor_approach/

[17] X. Michalet, AlliGator Software Manual (2025).
URL https://alligator-distribution.readthedocs.io/

[18] X. Michalet, AlliGator Software Github Repository (2025).
URL https://github.com/smXplorer/AlliGator

[19] D. Roy, X. Michalet, E. W. Miller, K. Bharadwaj, S. Weiss, Data associated with "Towards measurements of absolute membrane potential in Bacillus subtilis using fluorescence lifetime", Biophysical Reports, Volume 5, Issue 1, 2025, 100196 (2025). doi:10.6084/m9.figshare.28267028.v1.

[20] U. Schmidt, M. Weigert, C. Broaddus, G. Myers, Cell Detection with Star-Convex Polygons, Lecture Notes in Computer Science, Springer, 2018, book section Chapter 30, pp. 265–273. doi:10.1007/978-3-030-00934-2_30.

[21] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J.-Y. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, A. Cardona, Fiji: an open-source platform for biological-image analysis, Nature Methods 9 (7) (2012) 676–682. doi:10.1038/nmeth.2019.

[22] R. Ankri, A. Basu, A. C. Ulku, C. Bruschini, E. Charbon, S. Weiss, X. Michalet, Single-photon, time-gated, phasor-based fluorescence lifetime imaging through highly scattering medium, ACS Photonics 7 (1) (2020) 68–79. doi:10.1021/acsphotonics.9b00874.

[23] V. Pandey, I. Erbas, X. Michalet, A. Ulku, C. Bruschini, E. Charbon, M. Barroso, X. Intes, Deep learning-based temporal deconvolution for photon time-of-flight distribution retrieval, Optics Letters 49 (22) (2024) 6457–6460. doi:10.1364/OL.533923.

[24] Sphynx (2025).
URL https://www.sphinx-doc.org/

[25] Readthedocs (2025).
URL https://docs.readthedocs.com/

[26] C. Sherry, A. Verma, J. Smith, V. Pandey, T. Young, J. Williams, X. Michalet, X. Intes, M. Barroso, Near infrared fluorescence lifetime FRET microscopy to evaluate antibody drug binding in various HER2 positive cancer cell lines, Proceedings of SPIE 12384 (2023) 123840P. doi:10.1117/12.2654748.

[27] I. Erbas, V. Pandey, N. I. Nizam, N. Yuan, A. Verma, M. Barosso, X. Intes, Enhancing fluorescence lifetime parameter estimation accuracy with differential transformer based deep learning model incorporating pixelwise instrument response function, arXiv (2024).
URL https://arxiv.org/abs/2411.16896

[28] C. Sherry, L. Chavez, R. Kanai, S. Gao, X. Intes, M. Barroso, Multiplexed fluorescence lifetime imaging of near-infrared fluorescent proteins in 3D tumor spheroids, Proceedings of SPIE 12862 (2024) 1286208. doi:10.1117/12.3003280.

[29] N. Yuan, V. Pandey, X. Intes, X. Michalet, Experimental study of fluorescence lifetime uncertainty in time-gated iccd-based macroscopic fluorescence lifetime imaging, Optica Biophotonics Congress: Biomedical Optics (2024) TM5B.4doi:10.1364/translational.2024.Tm5b.4.

[30] N. Yuan, V. Pandey, A. Verma, J. C. Williams, X. Intes, M. Barroso, Antibody-target binding quantification in living tumors using macroscopy fluorescence lifetime Forster resonance energy transfer imaging (MFLI FRET), Proceedings of SPIE 12821 (2024) 1282105. doi:10.1117/12.3003282.

[31] N. I. Nizam, V. Pandey, I. Erbas, J. T. Smith, X. Intes, A novel technique for fluorescence lifetime tomography, bioRxiv (2025). doi:10.1101/2024.09.19.613888.

[32] V. Pandey, E. Millar, I. Erbas, L. Chavez, J. Radford, I. Crosbourne, M. Madhusudan, G. G. Taylor, N. Yuan, C. Bruschini, S. T. Radev, M. Barroso, A. Tobin, X. Michalet, E. Charbon, D. Faccio, X. Intes, Real-time wide-field fluorescence lifetime imaging via single-snapshot acquisition for biomedical applications, bioRxiv (2025). doi:10.1101/2025.04.16.646646.

[33] A. Verma, V. Pandey, C. Sherry, T. Humphrey, C. James, K. Matteson, J. T. Smith, A. Rudkouskaya, X. Intes, M. Barroso, A novel technique for fluorescence lifetime tomography, Advanced Science 12 (2025) 2403253. doi:10.1002/advs.202403253.

[34] X. Michalet, AlliGator v1.0.2 Release (2025). doi:10.5281/zenodo.15468169.