PyScanFCS
Data evaluation for perpendicular line scanning FCS
Software Guide

Paul Müller
Biotechnology Center of the TU Dresden
December 7, 2017

Contents

1 Introduction

1.1 Preface

PyScanFCS processes photon arrival times recorded in perpendicular scanning FCS setups using the Flex02-01D/C correlator from http://correlator.com.

The PyScanFCS software provided is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 2 of the License, or (at your option) any later version.

An exemplary usage of PyScanFCS - delivering a proof of principle from calibration to data fitting - was shown in [**?**] (Authors manuscript available at pyscanfcs.craban.de.

1.2 Prerequisites

In order to use the PyScanFCS software, the following equipment is needed:

Hardware

- A confocal laser scanning microscope (CLSM). For single-color one-focus SFCS, the CLSM should be capable of continuously scanning along a single line. For more sophisticated measurements, features like "Multitrack mode" for dual-color measurements and frame scanning for two-focus SFCS are required (LSM510, Zeiss, Germany).
- One or two avalanche photo diodes (APD) for single or dual color detection. In addition, matching filters and beam splitters. The detection channels are referred to as channels A and B throughout this documentation.
- A photon stream recording device connected to the APDs. In the tested setup a Flex02-12D/B correlator from correlator.com[1] was used in photon history recording mode. The device creates two binary files A and B that contain the photon arrival times detected by the corresponding APDs.

Software

PyScanFCS can be downloaded from http://pyscanfcs.craban.de.

- **Windows** For Windows, stand-alone binary executables are available from the download page.
- **Linux** There are executable binaries for widely used distributions (e.g. Ubuntu).
- **Sources** The program is written in python 2.7. On any operating system, PyScanFCS can be run from source. In general, the following packages need to be available:
  `cython (≥ 0.1.6)`
  `python-matplotlib (≥ 1.0.1)`
  `python-multipletau (≥ 0.1.4)`
  `python-numpy (≥ 1.6.2)`

[1] Note that for some reason the photon history recording mode does not work with the Flex02-01D/C correlator. Keep that in mind while troubleshooting your setup. When using a different device, the data has to be converted accordingly. The data format is described in section **??**.

`python-pyfits`
`python-scipy` ($\geq$ `0.10.1`)
`python-wxgtk2.8` (`2.8.12.1`)

There are two ways to make PyScanFCS run from source.
install from PyPI: `pip install pyscanfcs` and run the package directly `python -m pyscanfcs`.
clone from GitHub: Obtain the source directly from GitHub (https://github.com/paulmueller/PyScanFCS).

Cython PyScanFCS needs Cython to perform some CPU intensive operations. Before running PyScanFCS from source, a C compiler needs to be installed (http://docs.cython.org/src/quickstart/install.html).

Windows. The common MinGW compiler can be used. Instructions on how to install MinGW can be found in the Cython documentation: http://docs.cython.org/src/tutorial/appendix.html. On machines running Windows 7 x64 with the x64 version of python, the Microsoft Windows SDK for Windows 7 and .NET Framework 3.5 SP1 must be used:
http://wiki.cython.org/64BitCythonExtensionsOnWindows.

Ubuntu. The following packages need to be installed:
`build-essential`
`python-dev`

MacOS. Use Apple's XCode, available from http://developer.apple.com.

Cython can be obtained from http://www.cython.org/. After unpacking of the archive, Cython is installed by executing `python setup.py install` within the unpacked directory[2].

Correlation curves created by PyScanFCS can be processed using `PyCorrFit`[3] [**?**].

[2] Cython can also be installed via PyPI with `pip install cython` (http://pypi.python.org/pypi).
[3] http://pycorrfit.craban.de

2 Data acquisition modes

In order to achieve all possible measurement modes depicted in figure **??**, the CLSM needs to be set up properly. The scan paths should be checked for straightness e.g. by bleaching a layer of eGFP[4]. In 2fSFCS, the distance between the lines can be determined in a similar fashion. For each measurement, it is vital to record approximate scanning time and channel assignment. PyScanFCS attempts to find the correct scanning cycle, but manual intervention will be necessary in case of a failure[5]. This section describes the acquisition modes that

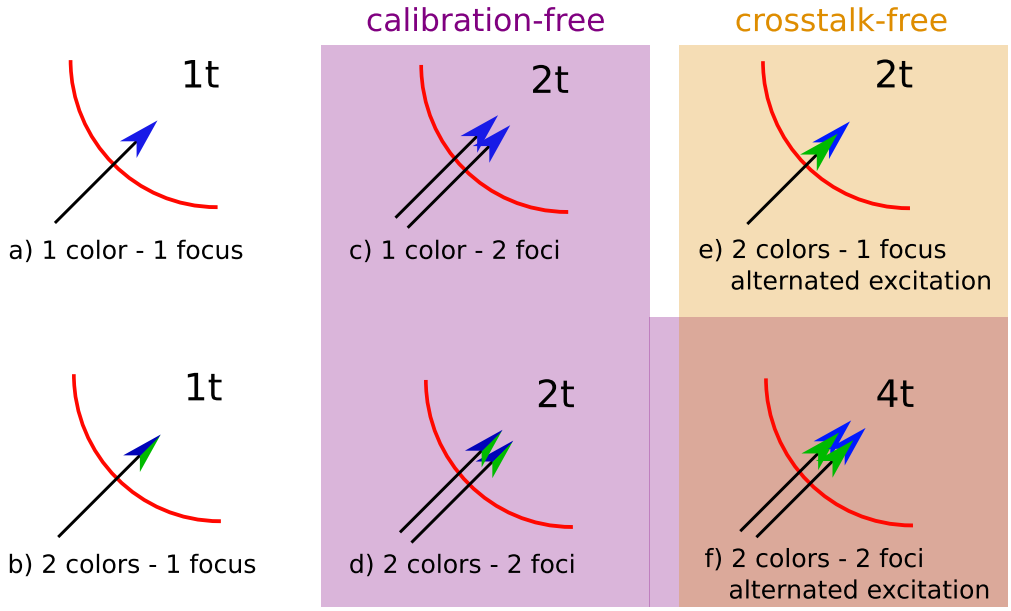


Figure 1, acquisition modes in PyScanFCS: The six figures **a-f** show possible scenarios for scan path and excitation settings. The alignment of the scan paths (black) is shown for membrane of a giant unilamellar vesicle (red). The arrow tips define the different colors used for each scan (blue excitation, green excitation, or both at the same time). Multiple arrow tips indicate the sequential scanning of one line with two colors. The time necessary to perform a complete scan in each scanning mode is shown respective to "one color – one focus" on the top right corner of each figure.

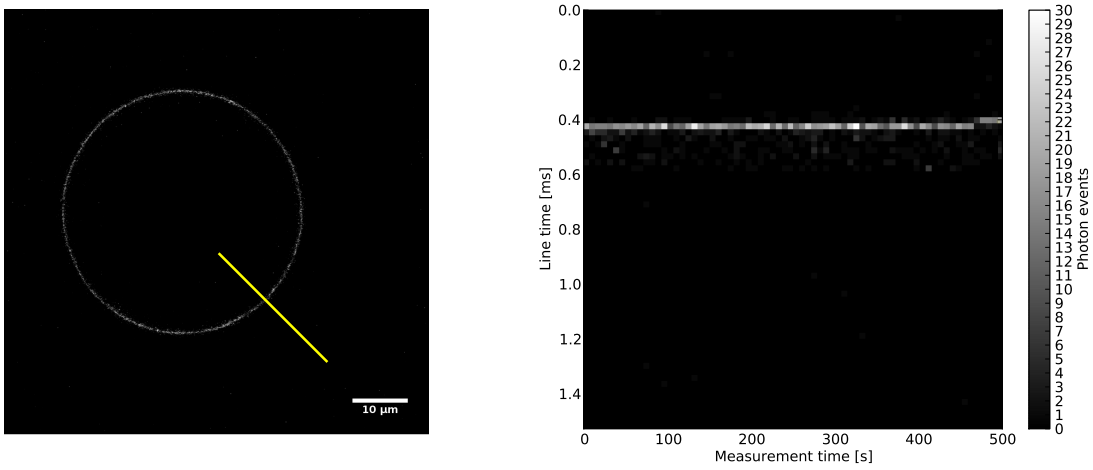
PyScanFCS supports. Correlation functions are labeled according to figure **??**. Figure **??** shows advantages and disadvantages in terms of time resolution, calibration and crosstalk.

a) one color - one focus One spot of the membrane is scanned continuously using one laser line. The signal is detected using one APD and PyCanFCS can be used to calculate one autocorrelation curve (AC_{A1}).

b) two colors - one focus The setup is the same as in (a), except that two laser lines are used and the signal is detected by two APDs. This mode can be applied for multiple dye studies. There are four possible correlations: AC_{A1}, AC_{B1}, CC_{A1B1}, CC_{B1A1}

c) one color - two foci The CLSM is set up to perform a line scan along two lines with a distance d apart. For detection, only one APD is required. The distance d needs to be

[4]Often, hysteresis effects occur during high speed scanning and scan paths are not straight. Switching to slower bidirectional scanning or setting up a "multitrack mode" might solve these problems.
[5]In some cases, the microscope software (AIM, Zeiss) might also calculate a wrong scan cycle time.



(a) GUV scan path in SFCS

(b) Kymograph produced by PyScanFCS

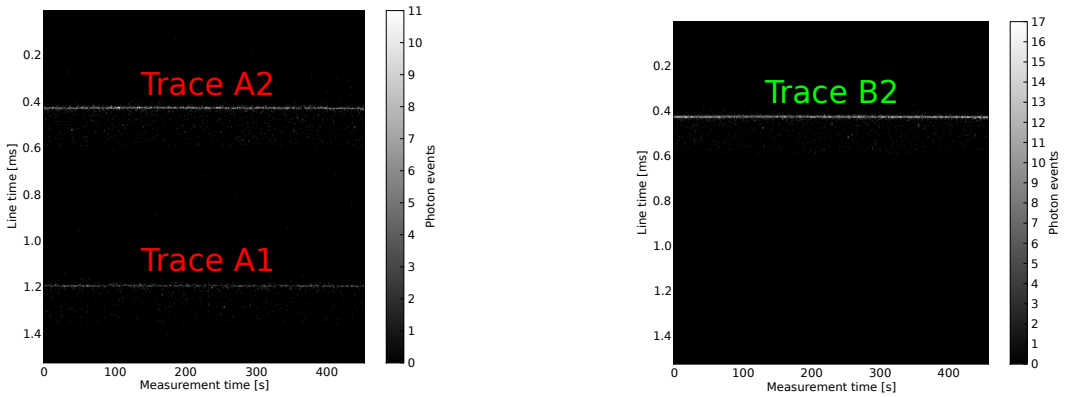
Figure 2, 1 focus 1 color measurement with PyScanFCS: Example of a GUV scanned using one focus and one laser line. **a)** The equator of a GUV with the scan path (yellow). **b)** A kymograph generated by PyScanFCS. The scan cycle time is 1.527 ms (vertical) and the measurement time is 500 s (horizontal). Each scan cycle is binned into 70 bins. Note that the resolution in direction of the measurement time is adapted to the scan cycle resolution. Also note the step at the end of the measurement. Such a step can be the result of a movement of the GUV during the measurement. This part needs to be excluded from the trace before calculating the correlation function.

determined beforehand. In addition to two autocorrelations, two cross correlation curves are obtained. The cross correlation curves contain information about the shape and size of the focus traversing the membrane. Thus, no calibration measurement with a reference dye is necessary to determine extents of the scanning focus. Possible correlations: AC_{A1}, AC_{A2}, CC_{A1A2}, CC_{A2A1}

d) two colors - two foci Like (b), this is a dual-color extension. Two laser lines are used for scanning and two APDs are used for detection.
Possible autocorrelations: 4
Possible cross-correlations: 12

e,f) dual color with alternating excitation To avoid cross-talk between the two color channels, as it likely happens in the *b) two colors - one focus* or *d) two colors - two foci* modes, the CLSM can be setup to sequentially scan the membrane with two laser lines. This way, only one laser line is used to excite the sample at a time and the fluorescence of two dyes can be separated, effectively circumventing crosstalk. The scan cycle time doubles when using alternating excitation and hence the binning time is twice as long. This effectively results in "loosing" data points[6] at the beginning of the correlation curve, which decreases the time resolution.
Possible correlations (1 focus): AC_{A1}, AC_{B2}
Possible correlations (2 foci): AC_{A1}, AC_{B2}, CC_{A1B2}, CC_{B2A1}

[6]For the multiple-τ algorithm used, exactly $m/2$ points are lost, where m is the number of points of the correlation curve that are calculated by the multiple-τ algorithm before the intensity trace is binned for the first time.



(a) A: Red channel

(b) B: Green channel

Figure 3, alternating excitation with PyScanFCS: The kymograph yields a scanning time for one line of 1.527 ms. One can clearly see the trace A2, which originates from crosstalk from the green dye to the red channel or from the red dye being excited by the green laser. There is no crosstalk from the red dye to the green channel, which in this case helps identify the traces.

3 Working with PyScanFCS

3.1 Data file formats

- **~.csv files** each contain a correlation curve. Further information is available in the `PyCorrFit` documentation (`http://pycorrfit.craban.de`).
- **~.dat files** are created by the software `Photon.exe` that ships with the correlators from correlator.com. The format stores the photon arrival time differences between two photon events. Data format is as follows[7]:
  - The value of the first byte (uint8) identifies the format of the file: 8, 16, or 32 bit.
  - The value of the second byte (uint8) identifies the system clock, usually 60 MHz.
  - The rest of the file contains the times between two photon arrivals. The time unit is 1/(system clock) (here $16.\overline{66}$ ns).
  - 8 bit format is not supported.
  - 16 bit format. Each word (2 bytes, uint16) represents a photon event. For low intensities, values of `0xFFFF` may occur. In this case, the following four bytes (uint32) represent a photon event.
  - 32 bit format. Each dword (4 bytes, uint32) represents a photon event. 32 bit files can be created from 16 bit files using `Photon.exe`. PyScanFCS saves ~.dat files in this format.
- **~.fits files** - The **f**lexible **i**mage **t**ransport **s**ystem is used by PyScanFCS to save already binned ~.dat files. Those files contain additional fits-keys:
  - `SysClck` - system clock in MHz
  - `Total` - total measurement time in system clock ticks
  - `Tcycle` - acquired scan cycle time in system clock ticks

[7]Note, that Windows uses the little endian byte order. This might lead to problems when using PowerPC.

- `Tbin` - binning of the photon events in system clock ticks
- `Binshift` - bins added to beginning of the entire 2D data array before plotting

Because these files can become quite large, it is not recommended to store the data in this file format. Also, information is lost due to the binning. It is possible to save a ~.fits file with only a few photon events binned (small file size), which contains the correct plotting parameters (cycle time, bin shift). This file can later be opened prior to opening the corresponding ~.dat file. This way, e.g. the scan cycle time does not have to be found again.

- **~.int files** are created during binning. They are straight 16 bit files that contain binned events from a ~.dat file.
- **~.txt files** are generated optionally, if the user wants to save the generated traces. The file contains photon numbers, separated by a new line. The binning time, as well as the corresponding intensities can be calculated from parameters given in the file.
- **~.zip files** are bundles of correlation curves (- stored as ~.csv files). They can be imported for fitting by `PyCorrFit`. ~.zip files are automatically stored in the working directory (location of ~.dat or ~.fits files).

3.2 The user interface

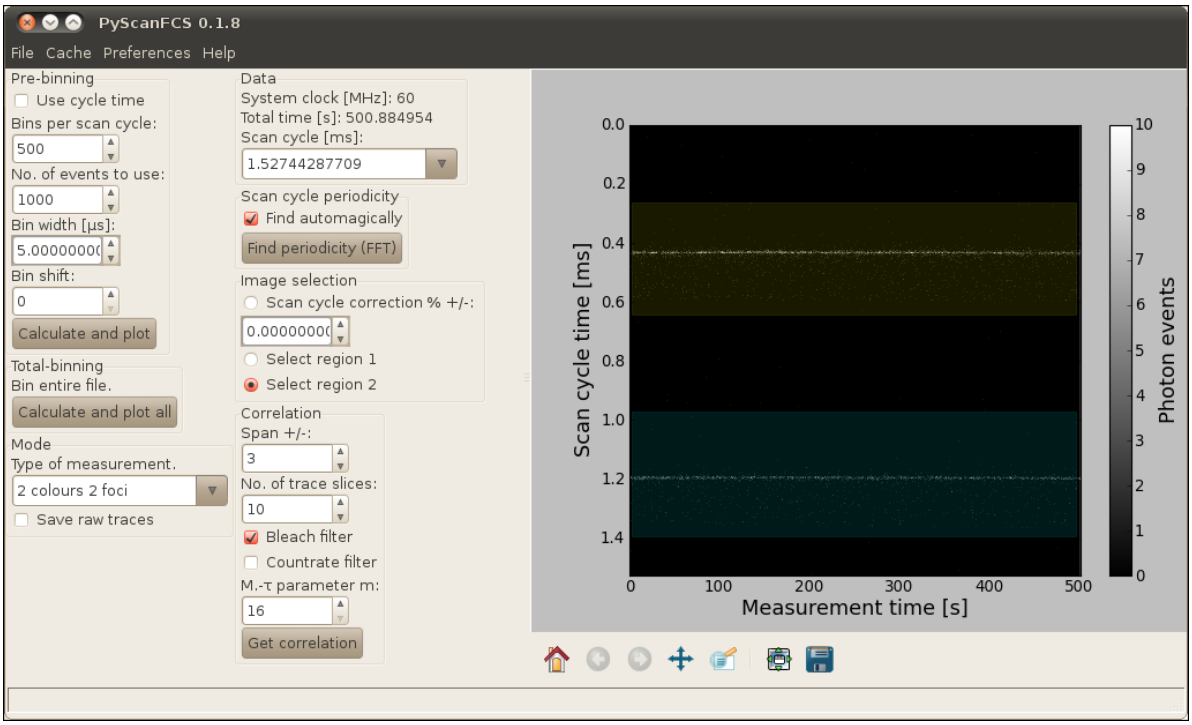


Figure 4, user interface: The left panel displays all parameters. The right panel shows the kymograph of the measurement.

Figure **??** shows the main user interface with the following content:

- The **menu bar** with standard operations, such as loading or saving a file. The **cache menu** displays all previously binned ~.dat files. The file menu contains a method to create artificial ~.dat files for testing.

- **Pre-binning**: This sets up all parameters that are used to bin a ~.dat file:
  - the binning resolution of the scan cycles
  - the number of events from a ~.dat file to process
  - the bin width of a scan cycle (not used if the scan cylce time is known)
  - a bin shift can be used to help selecting traces

  The bin shift parameter adds a number of empty bins to the beginning of the binned array. This will shift the vertical (scan cycle time) position of the visible trace.

- **Total-binning**: Bin the entire file using the scan cycle time found.

- **Mode**: Selects which mode to use. The modes are described above. Optionally save traces as ~.txt files inside the working directory.

- **Data**: Displays current file information and found scan cycle time. The scan cycle time may be doubled, quadrupled, halved or quartered. This may help, if the automagic algorithm did not find the correct scan cycle time.

- **Find line scan time**: Performs a fourier transform of the binned data[8]. If the check box is activated, the program will automatically find the correct maximum in fourier space and write back the corresponding scan cycle time. If the check box is not activated, a window will pop up for the user to define an interval where to search for a maximum. When a scan cycle time has been found, then the check box in **Pre-binning** will become active.

- **Image selection**: Selects which tool to apply to the binned image in the **kymograph**. This does not work, if any other tool (e.g. magnifier) from the kymograph toolbar is active.
  - scan cycle correction: This is used, if the scan cycle time found is not exactly correct (tilted lines in the kymograph). To correct the scan cylce time, the user draws a line along the trace in the kymograph. Afterwards, the kymograph can be recalculated using **Pre-Binning** or **Total-binning**.
  - Select region 1: Lets the user draw a rectangle around a trace that should be used for correlation. Region 1 is always displayed in yellow.
  - Select region 2: Depending on the `mode`, the user may select a second region for additional auto- and cross-correlation. Region 2 is always displayed in cyan.
  - Full measurement time: Maximizes the measurement time selection (horizontal in the kymograph) for the regions.

- **Correlation**: Calculates the traces and perform the multiple-τ algorithm according to the selected **mode**.
  - Span: The trace is calculated from the maximum of each scanned line. This corrects for spacial fluctuations perpendicular to the surface of the probed membrane. If the binning is too fine, or the scan cycle time is not exactly correct, this number can be increased to include more neighboring bins for the calculation of the trace.

[8]The user is advised to not use too many events. This might result in a system lock-up.

- No. of trace slices: Sometimes, dirt contaminates the trace. The trace can be split and the correlation curves can be calculated separately from each slice. `PyCorrFit` is able to average over user selected curves, yielding cleaner statistics.
- Bleach filter: Fits an exponential function to the selected trace and performs a bleaching correction according to [**?**]. Note that this bleaching correction is only valid for an approximately linear decay in the fluorescence signal. The bleaching data is exported to the zip file that also contains the correlation data. They are named

  `PyScanFCS_bleach_profile_TYPE_RANDOM.csv`

  where `TYPE` is the type of the curve (e.g. `Region_A1`) and `RANDOM` is just a random number.
- Countrate filter: Approximates the count rate of the fluorescence signal from the membrane. Because the laser focus is *scanned* through the membrane, the average signal from the measurement does not equal the actual count rate within the membrane. The algorithm fits a Gaussian to the average intensity profile and uses the standard deviation to calculate the residence time of the focus inside the membrane. This residence time is then used to calculate the approximate count rate.
- M.-τ parameter m: This is a parameter of the multiple-τ algorithm. m is the number of points of the correlation curve that are calculated by the multiple-τ algorithm in the first loop. Afterwards, the intensity trace is binned by combining two neighboring bins. Then, sequentially $m/2$ data points are calculated after each binning of the trace. A thorough description is given in [**?**]. $m = 16$ is a good measure.

- The **kymograph plot** shows the binned ~.dat files. The Pixel resolution in the direction of measurement time is adjusted to match the resolution of the scan cycle time (bins per scan cycle). This is implemented by slicing the array and showing only a few slices in measurement time "direction". So far, no disadvantages compared to averaging have been observed. The plotting area contains a useful toolbar.

Acknowledgements

I thank Eugene Petrov and Fabian Heinemann (MPI Biochem, Martinsried, Germany) for discussions on the implementation of the multiple-τ algorithm. I thank André Scholich (TU Dresden, Germany) for initial proof reading of the manuscript. The idea of a kymograph and the Fourier analysis method are adopted from a Matlab script written on the same issue by Jonas Ries.