Functional Ultrasound MATLAB object class code documentation

Ahmed El Hady[1,7,8*], Daniel Takahashi[6,*], Ruolan Sun [8], Tyler Boyd-Meredith[1], Yisi Zhang [1], Adam S Charles[4,5,*,+], Carlos D Brody[1,2,3,+]

1. Princeton Neuroscience Institute, Princeton University, Princeton, United States.
2. Howard Hughes Medical Institute, Princeton University, Princeton, United States.
3. Department of Molecular Biology, Princeton University, Princeton, United States
4. Department of Biomedical Engineering, John Hopkins University, Baltimore, United States
5. Mathematical Institute for Data Science, Kavli Neuroscience Discovery Institute & Center for Imaging Science, John Hopkins University, Baltimore, United States
6. Brain Institute, Federal University of Rio Grande do Norte; Natal, Brazil
7 Center for advanced study of collective behavior, University of Konstanz.
8 Max Planck Institute of Animal Behavior, Konstanz

+ correspondence should be addressed to Carlos D Brody (brody@princeton.edu) or Adam Charles (adamsc@jhu.edu)

* equal contribution.

This documentation outlines the basic functionality and methods implemented in the fUS analysis code-base. The code is written as a pair MATLAB object classes: one for single datasets, and one for analyzing groups of datasets. For a detailed example on the code's use, please see the `demo.m` script in the `code` folder.

1 fUS and fU-multi object classes

Two classes are defined in this code package: one for the exploration and processing of a single fUS movie, and one for multiple fUS movies taken over a sequence of experiments. The two classes (`uo` and `muo` for ultrasound object and multi-ultrasound object, respectively) provide a concise way of loading, viewing, and processing fUS data. The single data-set object `uo` contains most of the vital class methods and the multi-data set object class `muo` manages a set of `uo` object and handles the overhead of working with many datasets at once.

In initializing a single data-set `uo`, the data may either be loaded and then passed into the object,

```
>>fuo = uo(dataMatrix)
```

Alternatively a path to a `.mat` file can be provided, allowing the code to load data only as necessary:

```
>>fuo = uo([data/path/filename.mat])
```

This latter method of initializing an object is especially important when working with multiple datasets, as it lowers the memory requirements tremendously. Loading the data directly or setting pointers to the data to prevent the full data being loaded can be toggled by the 'loadToRAM' option as

```
>>fuo = uo([data/path/filename.mat], 'loadToRAM', true)
```

For more details on functions for data loading, see Table 1. To load multiple functional ultrasound datasets into one object, the uom object can be used as

```
>>fuom = uom([data/path/])
```

where the path containing multiple `.mat` files are located and can be identified and accessed. In the creation of the `fuom` object, each identified movie file in the path has a single `uo` object created in `fuom.uo`, along with reasonable meta-data. Functions run on fuom often use the `uo` class functions to efficiently run on all files more conviniently.

2 Data visualization

Built in to the uo and uom objects are methods to visualize parts or all of the data (see Table 2). For example, one can randomly select a number of traces to view (e.g., Figure 1) by selecting a subset of pixels (in this example 10).

```
>>fuo.displayExampleTraces('numTraces', 10)
```

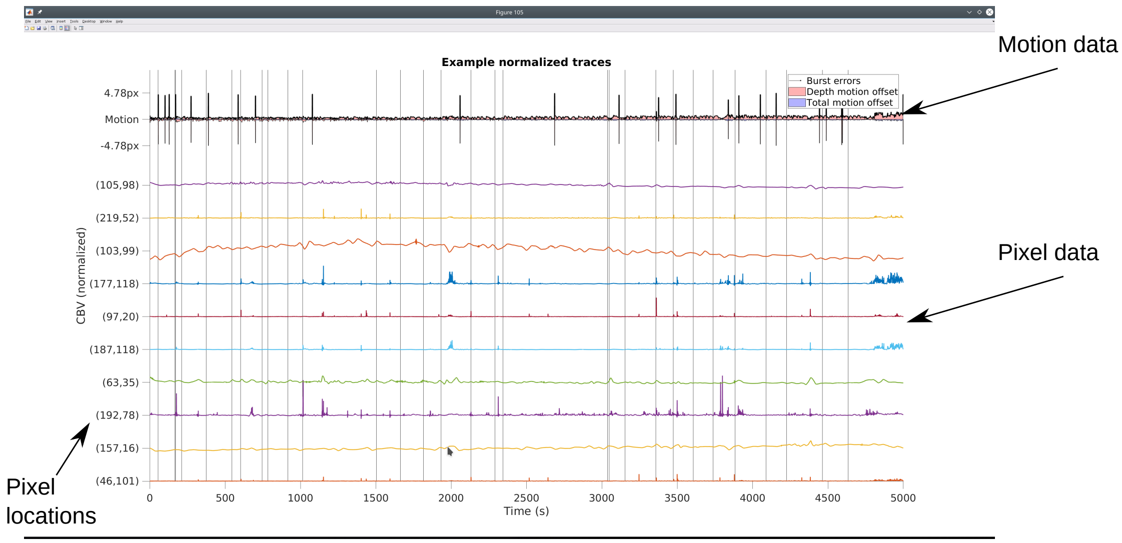

Figure 1: Example of displaying motion and time-trace data from an fUS movie.

To display the entire movie (e.g., Figure 2), the diplayMovie method can be used (NOTE: this method uses the MovieSlider package available at `https://github.com/sakoay/MovieSlider`)

```
>>fuo.displayMovie();
```

Single frames can also be displayed (e.g., Figure 3) via

```
>>fu.displayExampleFrame('frameNo','rand');
```

3 Motion and Error Correction

The first step in data analysis of fUS data is to determine where artifacts, in fUS primarily motion artifacts, exist in the data. Knowing which frames should be considered with lower confidence is vital to extracting information from the fUS recordings. The single fUS object class `uo` contains methods that isolate both burst errors and lateral rigid motion errors in the data.

Burst errors are defined as sudden, large increases in movie intensity over the entire field-of-view. Such events can be caused, for example, by sudden movements of the animal. Consequently, these events are simple to detect by analyzing the total intensity of the frames, as measured by the ℓ_2 norm $\sum_{ij} X_{ij}$ for

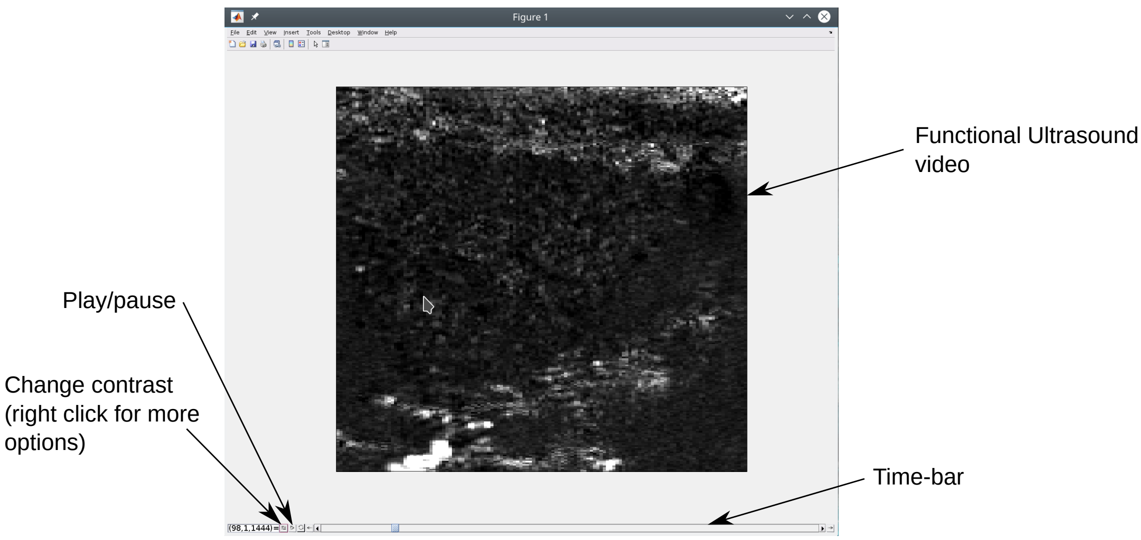

Figure 2: Example of single fUS movie display.

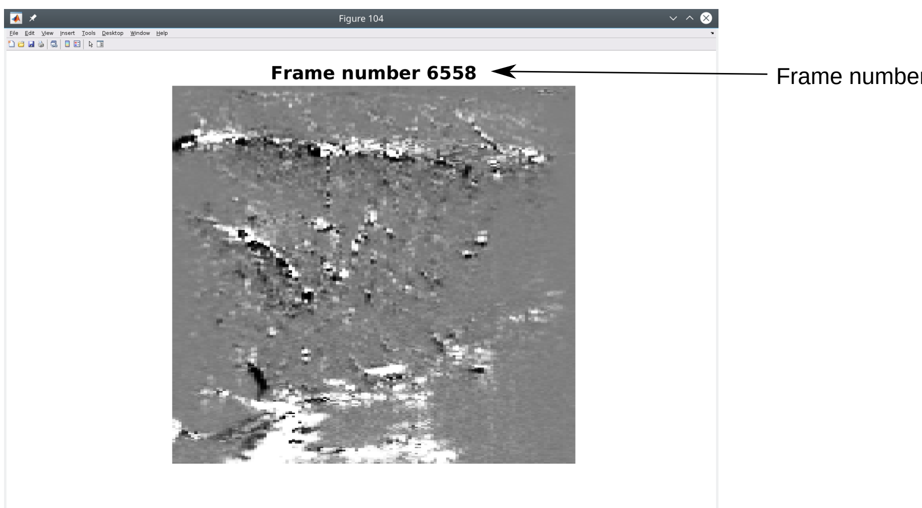

Figure 3: Example of single fUS frame display.

each frame X. We dynamically select a cut-off for determining a burst frame be estimating the inflection point in the histogram of frame-norms.

To ascertain the accuracy of the motion correction algorithm, we estimated the residual motion shift using a sub-pixel motion estimation algorithm based on fitting a Laplace function to the autocorrelation $C_{ij} = \langle X_{ref}, X * \delta(x-\tau_i, y-\tau_j)\rangle$, where X_{ref} is the reference image X is the image with the offset we wish to compute. The Laplace function is parameterized as

$$L(i,j;\rho,\mu_x,\mu_y\rho,\sigma_x,\sigma_y) = e^{-|\tau_i-\mu_x|/\sigma_x-|\tau_j-\mu_y|/\sigma_y}, \tag{1}$$

where and $\{\rho,\mu_x,\mu_y,\sigma_x,\sigma_y\}$ is the parameter set for the 2D Laplace function, including the scale, x-shift, y-shift, x-spread and y-spread respectively. We optimize over this parameter space more robustly by operating in the log-data domain, i.e.,

$$\arg\min_{\rho,\mu_x,\mu_y,\sigma_x,\sigma_y}\sum_{ij}\left[\log(C_{ij})-\log(L(i,j;\rho,\mu_x,\mu_y\rho,\sigma_x,\sigma_y))\right]^2 \tag{2}$$

$$=\arg\min_{\rho,\mu_x,\mu_y,\sigma_x,\sigma_y}\left[\log(C_{ij})-\log(\rho)-\frac{|\tau_i-\mu_x|}{\sigma_x}-\frac{|\tau_j-\mu_y|}{\sigma_y}\right]^2 \tag{3}$$

which makes the optimization more numerically stable, gradients are easier to compute and solving for $\log(\rho)$ directly removes one of the positivity constraints from the optimization. We further reduce computational time by restricting the correlation range to only shifts of $|\tau_i|, |\tau_j| < 25$ pixels and by initializing the optimization to the max value of the cross-correlation function $\{i,j\} = \arg\max(C_{ij})$.

Computing motion offsets relies on a global reference. If few frames are shifted, than the median image of the movie serves as a reasonable estimate. In some cases this is not possible, and instead other references can be used, e.g., the median image of a batch of frames at start or end of the video. We also provide the option to estimate the motion residual over blocks of frames, sacrificing temporal resolution of the estimate for reduced sensitivity to activity. In fUS, the temporal resolution sacrifice is minimal due to the initial round of motion compensation built into the image rendering.

Burst errors and rigid motion can be corrected via the code

```
>>fuo.findBurstFrames();
>>fuo.findMotionCorrectionError();
```

The first line identifies the burst frames and the second corrects rigid motion. The results of the rigid motion identification and location of burst error frames can be visualized next to the time traces using `fu.displayExampleTraces()`, e.g., see Figure 1. For more details on this functionality, see Table 3. You can also extract the inpainted frames using the command

```
>>[frameInPt,frameIDX] = fuo.computeBurstErrorInpainting();
```

4 Denoising

The `uo` and `uom` classes enable the application of time-domain denoising via wavelets.

```
>>fuo.denoiseTracesWavelet();
>>fuo.displayMovie('denoised', 'true');
```

`fuo.displayMovie` uses MATLAB's internal wavelet denoising either `wdenoise()` or `wdenoise()` and `cmddenoise()`, which can be selected using the `'wDenoiseFun'` parameter. We suggest using the default `wdenoise()` for speed.

5 Running GraFT on fUSi data

GraFT can be run using the following command:

```
>>fuo.fuGraFT('n_dict', 45, 'lamForb', 0.3, 'lamCont', 0.4, 'lamCorr', 0.3)
```

Which creates a substruct `fuo.GraFTout` that contains the spatial components `fuo.GraFTout.spatial` and the temporal components `fuo.GraFTout.TimeTrace`.

6 Class methods

When initializing a multi data-set `uom`, either a list of `.mat` file-names, a cell array of datasets, or folder path can be provided. In the former two, a series of `uo` objects will be created to match the datasets or filenames input. In the last option, the class methods will determine all `.mat` files in the folder path and its sub-directories, and create a set of `uo` objects to access those files. The methods for these classes allow for loading, viewing, and performing basic processing steps. In particular, pre-processing steps such as de-noising, error detection and error correction.

Data loading: Basic tools for loading data are outlined in Table 1.

Table 1: Class methods for data access

Function Name	Description
`uo()`	Main function to initialize and set up a fUS object
`makeGetBlockFunction()`	Creates a function that extracts a movie block from the full data
`makeGetMovieFunction()`	Creates a function that extracts the full movie data
`makeGetTraceFunction()`	Creates a function that extracts a single pixel time-trace from the full data
`isMatFileBased()`	Checks if a uo is mat-file based of if data is loaded to RAM
`ensureMask()`	Ensures that a mask has been provided to separate the in-brain pixels
`writeToAVI()`	Writes a dataset (pre- or post processing) to an AVI movie
`drawROI()`	Enables the user to draw a mask around the brain

Visualization: Being able to visually sort through data is paramount to understanding the neural recordings. A number of class methods focus on this aspect, permitting the viewing of specific frames, movie snippets, example time-traces, lateral motion estimation through time, etc. These functions are detailed in Table 2.

Table 2: Class methods for visualization

Function Name	Description
`displayMovie()`	Function to display a fUS movie using MovieSlider
`displayExampleFrame()`	Function to display an example frame from the fUS movie
`plotBurstErrors()`	Plot the burst errors to validate correct identification
`displayExampleTraces()`	function to display example time-traces from the fUS movie
`displayMotionError()`	Displays example motion errors visually

Motion and Error correction: Code to remove artifacts from imaging are detailed in Table 3.

Table 3: Class methods for data cleaning

Function Name	Description
`findBurstFrames()`	Function to identify frames with burst errors
`findMotionCorrectionError()`	Identifies rigid shift errors in fUS data
`ensureMotionErrsComputed()`	Ensures that the motion errors are computed for a given `uo`
`ensureMotionCorrection()`	Ensures that the movie for a given `uo` is motion corrected
`doesDenoiseExist()`	Checks if the denoised movie was already computed
`doesMotionCorrectedExist()`	Checks if the motion
`ensureDenoisedData()`	Checks if denoised data is available and if not denoises the data
`denoiseTracesWavelet`	Denoises a fUS movie one pixel at a time using wavelet denoising
`motionHypothesisTest()`	Tests the similarity of a time trace during large and small motion
`shuffleMotionHypothesisTest()`	Same as motionHypothesisTest but shuffling data
`getMotionVectors()`	Get the per-frame motion offsets
`getMotionCorrectedSize()`	Returns the correct size of the post-motion corrected movie
`compareMotionHypothesisTest()`	Compares the motion metric before and after motion correction
`computeBurstErrorInpainting()`	Fill in missing frames with bicubic interpolation
`correctResidualMotion()`	Shift frames to correct computed translational motion

Analysis: Basic analysis tools, including PCA, GraFT and correlation computations can be computed using the functions in Table 4.

Function Name	Description
`getBasicStats()`	Compute basic statistics per pixel and per movie
`getBaselineImage()`	Compute a baseline image to compare individual frames to
`calcBasicStats()`	Compute basic per movie statistics
`eventSTH()`	Compute an event-triggered average of the movie
`event2timeseries()`	Converts event times to a time-series spike train
`event2frame()`	Translate an event time to a movie frame
`ensureMedian()`	Ensure that the median image has been computed and is available
`fuGraFT()`	Apply GraFT to a fUS dataset
`fuPCA()`	Run PCA on a fUS dataset
`correlateMotionWithData()`	Correlate the motion estimates with individual pixel timetraces
`correlateToEvent()`	Compute each pixel's correlation with a cecurring event
`correlationMap()`	Compute a correlation map across the movie's spatial extent
`computeCorrsWithBaseline()`	Compute correlations of all pixels with a baseline time-trace
`clusterTraces()`	Cluster the pixels of the fUS data based on their time-traces
`alignTrials()`	Align all the trials based on event time-stamps

Table 4: Class methods for basic analysis

Extra functions: The ultrasound class is supported by a number of additional functions in Table 5, and a number of external functions from other sources, detailed in Table 6.

Table 5: Support functions

Function Name	Description
`applyMask()`	Applies a user-defined mask to a frame or video of the data.
`extractPixel()`	Extracts N pixel time traces from the ultrasound movie
`findSigmoidPars()`	Fits a sigmoid to data
`fit2DLaplaceFun()`	Fits a 2D laplace function to data
`gaussfilt()`	Filters with a gaussian kernel
`greedyEND1D()`	Greedy solver for a 1D earth-mover's distance
`histogramDistance()`	Computes a distance between histograms
`lapFunc()`	Computes a laplace function in 2D given data and parameters
`lapLogFunc()`	Same as lapFunc but computation is in the log-domain
`motionFitSingleTest()`	Compare a single frame to a reference to identify translational motion
`plotAllAnats()`	Plots all the anatomical images for 4 different sessions
`plotWaveImages()`	Plots wavefronts of waves
`pmColorMap()`	Sets a color map with one color for pos. to a different color for neg.
`realignSingleFrame()`	Linearly translates a frame given a shift
`robustTwoSidedSTD()`	Computes a robust two-sided standard deviation
`selectFusiTraces()`	Selects a subset of time-traces (single pixels) from an ultrasound video
`softScale()`	Scale values of an array smoothly given a range to keep linear

Table 6: External functions

Function Name	Description
`AdvancedColormap()`	Function that creates additional color maps for plotting
`distinguishable_colors()`	Generates maximally distinguishable color sets for plotting
`halfSampleMode()`	Computes an approximate mode of a distribution
`robustSTD()`	Computes the robust standard deviation of a distribution

Multi-session data: Functions for multi-session data are described in Table 7.

Table 7: Functions for the `uom` object class

Function Name	Description
`uom()`	Main function for creating a uom object.
`ensureAllMasks()`	Makes sure all sub-objects have user-defined brain selections
`denoiseMulti()`	Denoise all uo datasets
`findMotionCorrectionErrorAll()`	Find translational motion for all datasets
`multiEventCorr()`	Correlate an event for all datasets
`multiMotionCorrect()`	Correct motion in all datasets
`multiMotionTest()`	Test motion correction in all datasets
`removeTrivialDatasets()`	Aux function to remove small datasets not worth analyzing