Little Objects Segmentation & Tracking Environment

Contents

Overview

LOBSTER is an image analysis environment designed to batch process multidimensional microscopy images; it is meant to identify biological objects and measure their location, co-location, morphology, dynamics and intensity distribution. The software was designed with speed and simplicity in mind and with a strong emphasis toward results validation and exploration. Some typical applications are:

- Cell phenotyping in high-content screening assays
- Objects tracking in microscopy time-lapses
- Filaments tracing / spots counting in large 3D images (e.g. lightsheet microscopy)

LOBSTER runs in MATLAB environment. The software deployment is extremely simple and no MATLAB programming knowledge is required to use it and customize image analysis workflows. Step by step tutorials covering real image analysis scenarios are provided to get started and can be used as templates to analyse slightly different images.

Why LOBSTER?

A growing number of biological images analysis software are available, but most suffer from at least one of the following shortcomings:

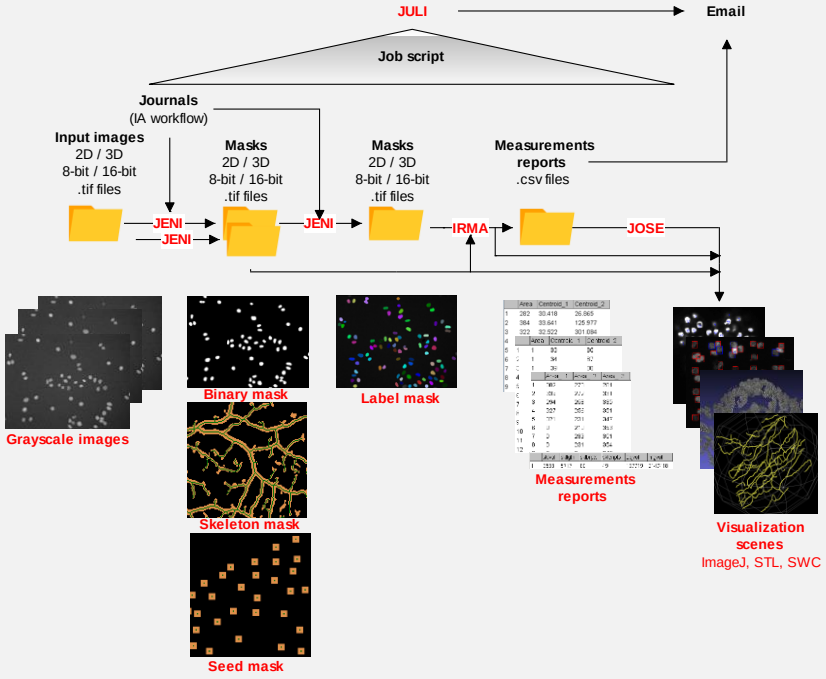
1) Too generic: many algorithms, no guidance to assemble them into practical workflows
2) Too specific: 2D images only, no object tracking, etc.
3) Not scalable: limited image size, slow, useless if the image quality degrades locally
4) Not flexible: fixed workflows, no customizable measurements
5) Not expressive: too much code to write, prone to error

LOBSTER attempts to strike a good balance between these needs and offers:

- A consistent set of documented, high level, image analysis functions
- A flexible framework to assemble them into practical workflows
- Many pre-configured workflows and sample images demoing real applications
- Many tools to explore and export results
- An analysis server to queue and process jobs and process them sequentially.

A layered framework

LOBSTER architecture decouples objects identification, measurements and visualization. These operations are respectively performed by three computational modules: **JENI** (identify objects), **IRMA** (object measurements) and **JOSE** (export results as scenes).



TIFF images from an input folder are processed into mask images by **JENI**, applying a sequence of image analysis functions (workflow) to identify biological objects. The workflow is configured in a journal, a human readable text file setting input/output folders, the sequence of functions and their parameters. Depending on the input images, there are three types of journals: 2D (.jl), 3D (.jls) and time-lapse (.jlm). Time-lapse journals can also process 3D images (extension is still .jlm). The output masks can be of four kinds:

Binary mask: connected groups of pixels delineating plain objects
Skeleton mask: 1-pixel curved segments (e.g. marking filament centrelines)
Seed mask: spots (e.g. marking the centre of detected objects)
Label mask: connected groups of pixels delineating plain objects (unique object IDs)

Several channels can be processed, for instance to perform intensity measurements inside identified objects. Masks are valid inputs to other journals, for instance when performing object tracking, co-localization or sub-counting (e.g. count spots inside objects). Large 3D images not fitting in memory are automatically chunked into sub-volumes to enable their processing (3D image bricking). Objects measurements (geometry and intensity) are performed by **IRMA**; all measurements are exported to .csv reports. For review/exploration, identified objects can be exported as 3D models (**IRMA**), or displayed as regions of interest in ImageJ (**JOSE**). Calls to the modules can be performed interactively from MATLAB console or be recorded to job scripts (**.m** text files) to simplify their further deployment. Finally, job scripts can be queued to **JULI**, an image analysis server able to email results upon job completion.

Requirements

Core: MATLAB (tested: 2015a) + Image processing + Statistics & Machine Learning toolboxes

Deep learning (optional): Parallel computing toolbox

Results exploration (optional)**:** ImageJ (recommended version: Fiji Lifeline June 2014)

3D image renderer (optional): Windows + Microsoft .NET framework 4 (installer provided)

It is highly recommended to run **LOBSTER** on an x64 Windows multicore workstation with at least 16 GB main memory (Windows 7/8/10/Server 2012/Server 2016). Pre-compiled functions and batch scripts are only provided for Windows and need to be compiled for other platforms.

Installation

Assuming Matlab is installed, download (green download button) **LOBSTER** archive from: https://github.com/SebastienTs/LOBSTER, and unzip content to an empty folder where **MATLAB** has read/write access. The path to this folder will now on be referred to as **ROOT_LOBSTER**. Since **LOBSTER** includes many sample images, they should be downloaded from an external source: create an empty folder named *Images* in **ROOT_LOBSTER** and unzip the content of this archive. Finally, create an empty folder called *Results* in **ROOT_LOBSTER**, download all results from **LOBSTER** sample workflows here and unzip content to *Results* folder.

Windows x64 (recommended)

For 3D renderer support, install Microscoft .NET framework 4 (provided here as reference in case unavailable from original source).

Linux / MacOs (3D renderer not supported, compiled C/C++ files not provided)

Matlab compiler is required and, for best performance, should be configured with a compiler supporting OpenMP (e.g. GCC). After initialization (see startup), type “compile” in MATLAB command line (only before first use).

Additional tools

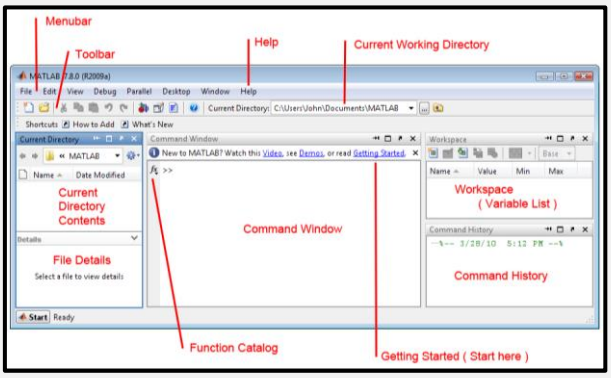
To visualize STL meshes: http://www.meshlab.net/. To visualize SWC filament networks / object tracks: https://www.reading.ac.uk/neuromantic/html/body_download.html.

Both can be installed in LOBSTER_ROOT/Tools for convenience.

Optional: If you plan to use **LOBSTER** server (not recommended to get started), download this archive and unzip the content in an empty folder where MATLAB has read/write access. The path to this folder will now on be referred to as **LOBSTER_sandbox**. The files of this archive include server ready job scripts.

Startup

1) Launch / configure MATLAB



MATLAB starts up in a default working directory (top banner). To set current working directory to **LOBSTER_ROOT**, enter the following line in command window (a.k.a. console) after replacing the string **'E:/LOBSTER'** by the path to **LOBSTER** installation folder (on your machine):

>> cd(**'E:/LOBSTER'**)

2) Initialize LOBSTER >> init

Alternatively, on Windows systems, you can double click **ROOT_LOBSTER**/Tools/init.bat or create a shortcut to this file. This will automatically launch Matlab and initialize **LOBSTER**. For this latter option you should first edit the file init.bat (notepad), and set the path to **LOBSTER** installation folder on the first line.

MATLAB basics

Only some basic understanding of MATLAB variables and functions is necessary to follow the tutorials of this document, this information is summarized below.

- Assign numerical value to variable '**a**': >> a = 5.6
- Use an existing variable in an expression: >> a+1
- Call a function by passing an existing variable: >> floor(a)
- Get help on a function: >> help floor
- Assign text string to variable '**c**': >> c='E:/LOBSTER'

To recall previous calls, press upper/lower keyboard arrows to scroll through history.

EXAMPLES

JENI (Journal ENgine Interface)

1) After initializing **LOBSTER**, enter >> JENI in MATLAB console
2) From file browser, select journal **'Tissue_SegWaterTiles.jl'** (in subfolder **'jl'**)
3) An image viewer showing an image with overlaid segmented objects pops up
4) Press '**m**' to toggle mask overlay, press '**x**' to process next image
5) Journal execution stops when all images from input folder have been processed. You can also abort it before by pressing '**q**'.

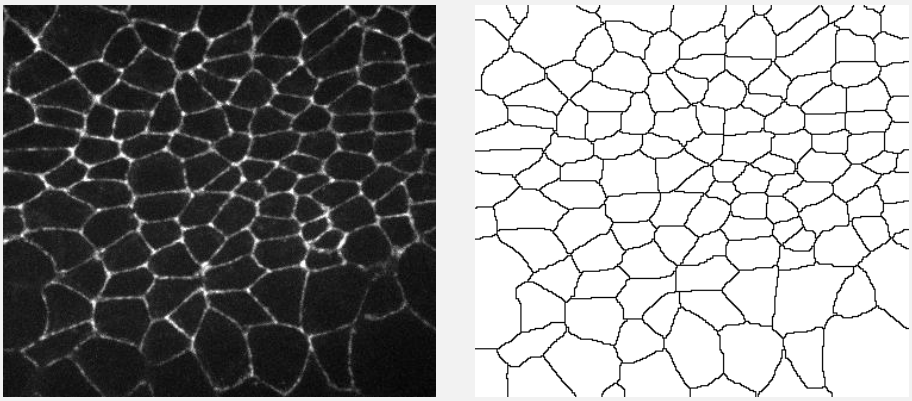
To call a journal without selecting it from file browser: >> JENI('Tissue_SegWaterTiles.jl');

Input and output (image) folders are configured in the journal itself; after launching a journal these folders are accessible from console blue hyperlinks.

```
>> JENI('Tissue_SegWaterTiles.jl');
Journal: E:\LOBSTER/Journals/jl/Tissue_SegWaterTiles.jl --> Launch
Input Folder: ./Images/Tissue/
Output Folder: ./Results/Images/Tissue/
```

In this example, output images are binary masks encoding segmented cells surrounded by zero pixels. Open the journal file in MATLAB editor (journal hyperlink) and edit cell detection sensitivity by lowering the parameter *ExtendedMinThr* from 5 to 1. Save the journal (Save icon) and re-launch it (**Launch** hyperlink): many cells are now erroneously fragmented. Restore cell detection sensitivity to its original value in the journal by the same procedure.

Note: Paths starting by `'./'` are relative to **LOBSTER_ROOT**.



Input image (left) and corresponding output image (right)

IRMA (Image Regions Measurements & Analysis)

Launch the previous journal, this time keeping its input/output folder paths to Matlab variables (arbitrarily called InputFolder and **MaskFolder**):

```
>> [InputFolder MaskFolder] = GENI('Tissue_SegWaterTiles.jl');
```

Note: We used GENI instead of JENI to prevent image display, processing is also quicker.

Perform basic object measurements by processing output (**MaskFolder**) from previous journal:

```
>> IRMA(MaskFolder, '', 'Objs', 2);
```

Only a measurement summary is displayed to console. Use second argument **'.'** to export complete measurement reports to file (at default location):

```
>> IRMA(MaskFolder, '.', 'Objs', 2);
```

Reports are quickly accessible from console hyperlink and can be opened in any data analysis software (or even ImageJ!). For 'Objs' (plain objects), basic measurements include:

- **Objects areas**
- **Objects centroids**
- **Objects bounding boxes**

	Area	Centroid_1	Centroid_2	BoundingBox_1	BoundingBox_2	BoundingBox_3	BoundingBox_4
1	282	30.418	26.865	21.500	15.500	20	24
2	384	33.641	125.977	21.500	113.500	23	24
3	322	32.522	301.084	22.500	290.500	21	21
4	321	36.287	69.897	23.500	59.500	28	21
5	384	36.641	203.977	24.500	191.500	23	24
6	396	43.801	154.449	31.500	143.500	23	23
7	396	46.801	232.449	34.500	221.500	23	23
8	398	52.731	82.638	36.500	69.500	30	24
9	288	54.837	278.760	42.500	269.500	24	17

2D objects measurements: area (pix), centroid XY coordinates (1: X, 2: Y) and bounding boxes (1: upper left X, 2: upper left Y, 3: width, 4: height)

IRMA arguments (basic measurements)

1) Path to **MaskFolder** (output folder from **JENI**)
2) Path to report folder (**''** no report, **'.'** default folder*)
3) Type of objects (**'Objs'**, **'Spts'**, **'Skls'**, **'Trks'** or **'Spst'**)
4) Image spatial dimensions (2 or 3)

*default folder is **LOBSTER_ROOT/Results/Reports/MaskFolder**

3D image example

Run the following journal demoing blood vessels tracing in 3D images (.jls journal):

```
>> [InputFolder MaskFolder] = JENI('BloodVessels3D_LocThr3DSkl3D.jls');
```

A slice browser pops up showing input images and coloured skeleton overlay: skeleton end points are blue (level 220), branch points are cyan (level 250) and regular skeleton points yellow (level 200). Scroll through slices with mouse wheel; adjust intensity by holding left click and moving mouse. While moving over a pixel, its intensity value is displayed in lower left corner (both for the image and the mask overlay). To enhance the continuity of the filaments, the depth of the local Z-projection (a "see through" effect applied to mask images) can be adjusted by pressing **z**. In the projection, yellow pixels are in the current slice, and blue pixels in front/behind. This journal configures the default local Z-projection to 0 slice, this value can be edited in the journal (variable RunProj). To help reviewing the results, 3D rendering can be performed on a sub-volume by pressing **r** and drawing a bounding box. The depth of the volume is set by the current projection depth.

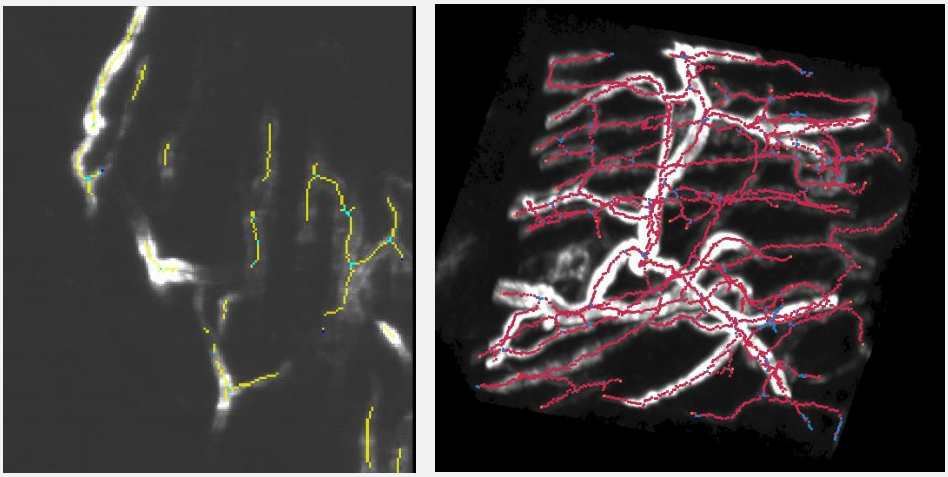
Now, perform blood vessel skeleton measurements:

```
>> IRMA(MaskFolder, '', 'Skls', 3, 3);
```

For 'Skls' (skeletons), basic measurements include (in this order):

- **Skeleton volume** (vox)
- **Skeleton linear length** (pix)
- **Skeleton number of branch points** and **end points**
- **Object volume** (vox), number of non-null voxels in mask
- **Image volume** (vox)

3D rendering can be enabled (Windows) by setting variable **Shw** to **5** in the journal. In 3D viewer, rotate view by holding left click, zoom in/out by holding right click, pan view by holding middle click. Toggle overlay with '**m**', adjust image contrast by pressing '**c**' and setting maximum intensity clipping to **<1**. **Note**: The dialog box can be hidden behind the renderer!



Slice browser (left) and 3D rendering (right) of input 3D image with skeleton overlay

3D Image: Blob/spot detection

The following journal detects nuclei in a 3D image:

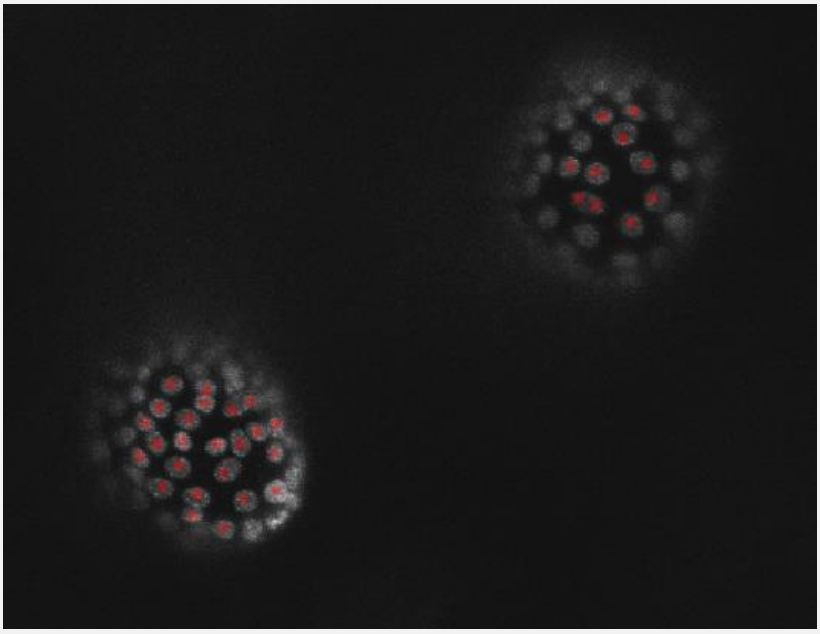
```
>> [InputFolder MaskFolder] = JENI('TissuePilar3D_NucDet.jls');
```

In slice browser, zoom in/out by holding middle click + moving around. Pan view by holding right click + moving around. To perform spot measurements:

```
>> IRMA(MaskFolder, '.' , 'Spts' ,3);
```

Spot count is displayed in console.

Basic measurements report spot locations (**X, Y, Z coordinates**).

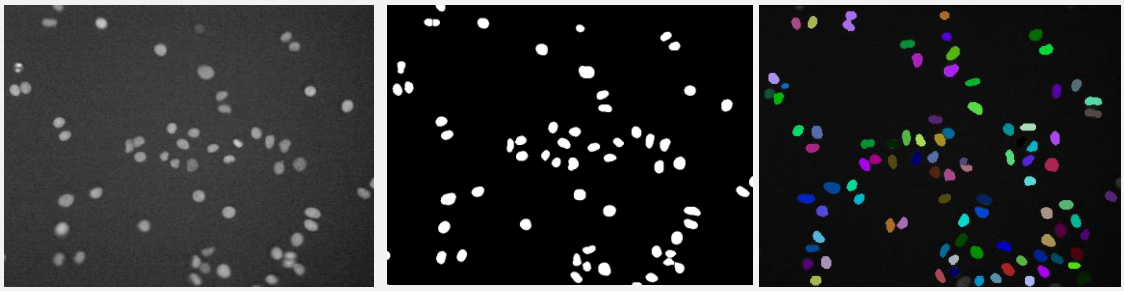


Detected nuclei (slice view)

Sequencing two journals: 2D time-lapse example

```
>> JENI('HeLaMovie_LapThrBinWatTiles.jl');
```

This journal processes a sequence of 2D images (folder of images) into a binary mask sequence (folder of images, segmented nuclei).



Original image (left), binary mask (middle) and label mask (right)

The next journal processes the object masks and performs object tracking:

```
>> [InputFolder MaskFolder] = JENI('HeLaMCF10A_TrackOvl.jlm');
```

If you inspect these journals, you will notice that the input folder of the second journal is set to the output folder of the first journal:

```
InputFolder = './Images/HeLaMCF10AMovie/Movie1/';
OutputFolder = './Results/Images/HeLaMCF10AMovie/Movie1/';
```

Excerpt from first journal 'HeLaMovie_LapThrBinWatTiles.jl'

```
InputFolder = './Results/Images/HeLaMCF10AMovie/Movie1/';
OutputFolder = './Results/Images/HeLaMCF10AMovieOvlLbl/Movie1/';
```

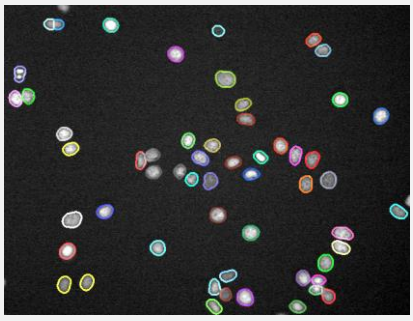
Excerpt from second journal 'HeLaMCF10A_TrackOvl.jlm'

When calling a .jlm journals (time-lapse), all images from input folder are processed **conjointly** into a label mask sequence where objects IDs are encoded by unique grayscale levels (lower left values when hovering over the objects).

To measure tracked objects, close the slice viewer and type:

```
>> IRMA(MaskFolder, '.', 'Trks', 2);
```

For 'Trks', basic measurements consist in **object areas** and **objects centroids**, reported in different files. To check the results together with the original images, you can conveniently run the macro **LOBSTER_ROOT/Tools/OverlayTrackedObjects.ijm**. Refer to appendix **LOBSTER** tools for usage.



Tracked objects results overlaid on original images

Intensity measurements

Call the following journal segmenting clustered nuclei:

```
>> [InputFolder MaskFolder] = JENI('NucleiCytoo_GradWaterTilesMerge.jl');
```

In this example, **InputFolder** holds several channels but the journal is configured to process only files with text string '**C00**' in their name (nuclei channel). See section "Dissecting a 2D image journal" for details.

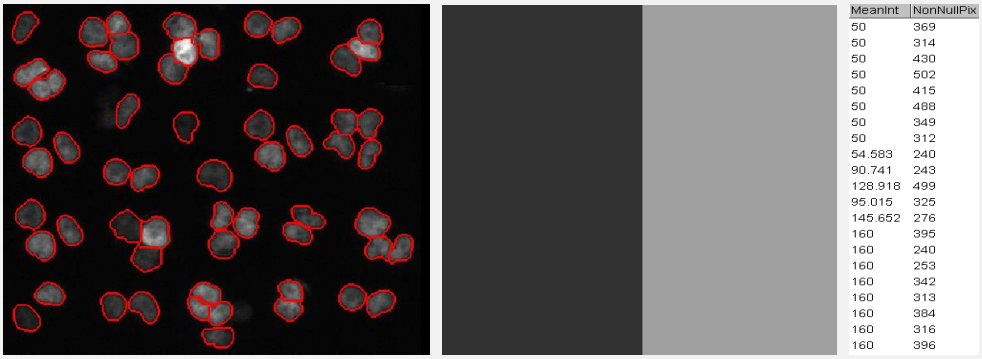
Measure average intensity inside segmented nuclei in channel '**C01**':

```
>> IRMA(MaskFolder, '.', 'Objs', 2, 1, InputFolder, '*C01*.tif');
```

To measure intensity in several channels (up to 3):

```
>> IRMA(MaskFolder, '.', 'Objs', 2, 1, InputFolder, '*C01*.tif', '*C02*.tif');
```

Intensity measurements are written to columns **MeanInt_X,** where **X** is the channel index (in the same order as they appear in **IRMA** call).



MeanInt	NonNullPix
50	369
50	314
50	430
50	502
50	415
50	488
50	349
50	312
54.583	240
90.741	243
128.918	499
95.015	325
145.652	276
160	395
160	240
160	253
160	342
160	313
160	384
160	316
160	396

Channel 'C00' (left) and corresponding channel 'C01' (middle) displaying a dark region (50) and a light region (160). Measurements report (right)

IRMA optional extra arguments (intensity measurements)

5) Image Z aspect ratio (default: 1) / model mesh exportation
6) Path to folder with intensity measurements images (**ChanFolder**)
7) Channel images filters (text strings, separated by comas)

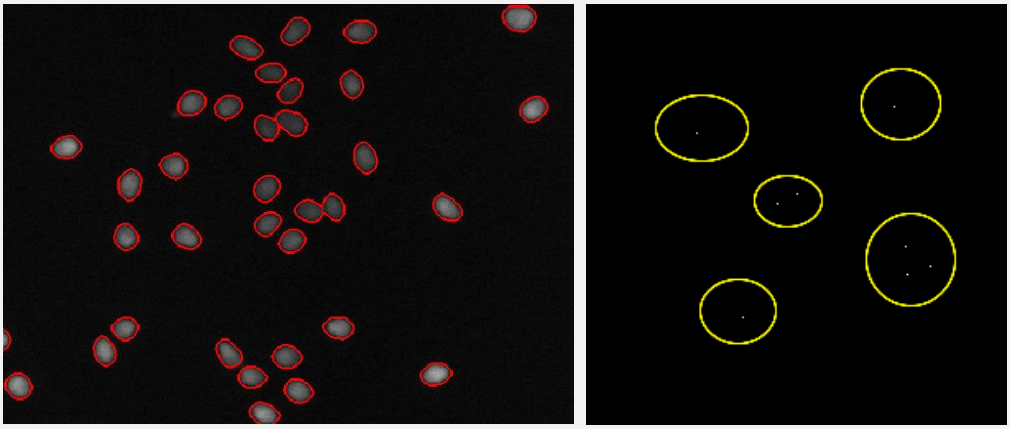
Note: **ChanFolder** can be set to {ChanFolder1, ChanFolder2}, in which case different folders correspond to the channel filters (there must be as many ChanFolder as channel filters).

There **must** be as many images checking the channel filter in **ChanFolder** as images in **MaskFolder**!

Objects co-measurements

Counting spots inside nuclei (FISH)

The results from several journals can be combined to perform object co-measurements; for instance the following example segment nuclei in a channel, detect spots in another channel and reports the number of spots (FISH) per nucleus.



Nuclei segmented by 'FISH_nucseg.jl' (left), scheme showing the principle of object co-measurements to count spots (markers) inside nuclei

First, segment nuclei in DAPI channel:

```
>> [InputFolder1 MaskFolder1] = JENI('FISH_nucseg.jl');
```

Then, detect spots (processing a FISH channel):

```
>> [InputFolder2 MaskFolder2] = JENI('FISH_sptdet.jl');
```

The result is a binary image with zero background and bright pixels (grayscale level 200) marking spots. Finally, measure the nuclei segmentation mask with spot seed mask set as **intensity channel**:

```
>> IRMA(MaskFolder1, '.', 'Objs', 2, 1, MaskFolder2);
```

In the reports, the column **NonNullPix** encodes the number of non-zero pixels in intensity channel; that is exactly the number of detected spots per nucleus!

In input folder, there are 3 FISH channels available (C01, C02 and C03). To process another channel you just have to edit journal '**FISH_sptdet.jl**' and change the image filter (line 4).

Combining journals: Job scripts

Journals are primarily meant to identify a single type of objects but several journals can be sequenced to identify different objects. This can be achieved interactively from the console as was shown in previous sections, or by running job scripts (.m files). Job scripts are MATLAB script text files storing console commands that are executed line by line. Sample job scripts are stored in **ROOT_LOBSTER\Jobs**.

For instance, for the FISH example:

```
[InputFolder1 MaskFolder1] = GENI('FISH_nucseg.jl');
[InputFolder2 MaskFolder2] = GENI('FISH_sptdet.jl');
IRMA(MaskFolder1,'.','Objs',2,1, MaskFolder2);
```

Job script FISH.m

Here GENI was used instead of JENI to prevent image display and user interaction.

To run a job script located in **ROOT_LOBSTER\Jobs**, simply enter its name:

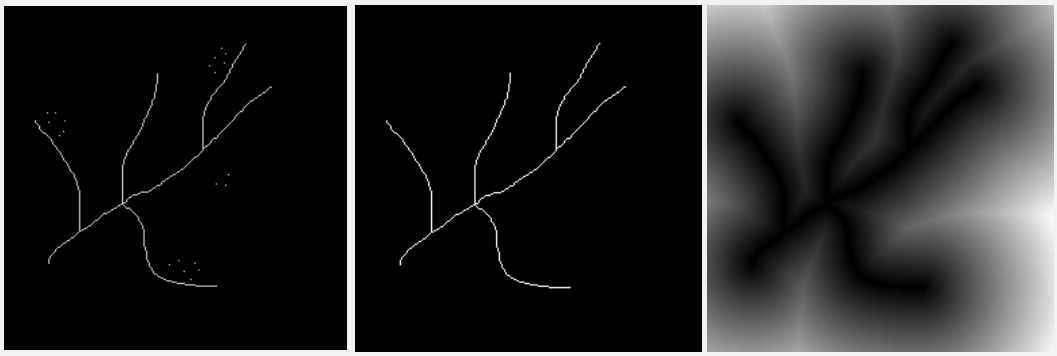
```
>> FISH
```

Note: To edit a script, browse to its location and double click corresponding *.m file.

Some more objects co-measurements examples

Distance distribution of spots / skeleton (synthetic image)

Distance map transforms a binary mask in a grayscale image where the grayscale value of each pixel encodes the distance to the closest object pixel. Distance maps can be automatically exported by setting variable **ExportDist = 1** in 2D or 3D journals.



Original image (left), filament skeleton (middle) and filament distance map (right)

The following journal segments filaments in a synthetic image and exports distance map:

```
>> [InputFolder1 MaskFolder1] = JENI('VesselsSpots_vesseg.jl');
```

This journal detects spots in the same image:

```
>> [InputFolder2 MaskFolder2] = JENI('VesselsSpots_sptdet.jl');
```

Now, the spot to filament distance distribution can be estimated by measuring spot intensity in the distance map:

```
>> IRMA(MaskFolder2, '.', 'Objs', 2, 1, MaskFolder1, '*dst*');
```

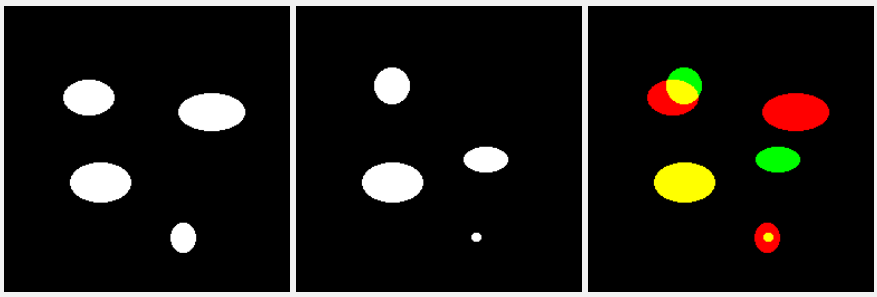
The distance from spots to the closest filament is encoded as spot intensity in the reports.

Distance map can also be computed **inside** objects (set **ExportDist = 2** in journal). For instance, the job script **FilamDiam.m** shows how to estimate the diameter of blood vessels by computing distance map inside segmented filaments and measuring intensity values over the vessel centerlines (skeleton).

Note:

- Distance map images are appended **'_dst'** to their filenames; these images are automatically ignored by IRMA in **MaskFolder**.

Objects colocalization (synthetic images)



Binary mask of objects identified in first (left) and second channel (middle), overlay showing partial, complete or no co-localization (right)

Call two journals segmenting objects in synthetic images (2 channels):

```
>> [InputFolder1 MaskFolder1] = JENI('ObjectsColoc_C00.jl');
```

```
>> [InputFolder2 MaskFolder2] = JENI('ObjectsColoc_C01.jl');
```

Measure the overlap between both masks with:

```
>> IRMA(MaskFolder1, '.', 'Objs', 2, 1, MaskFolder2);
```

In reports, the column **NonNullPixOvl** encodes the number of non-zero pixels in intensity channels; dividing these numbers by object areas brings the normalized object overlaps!

Montaging identified objects

To quickly review thousands of localized objects detections spread across a large 3D volume, it can be convenient to use the ImageJ macro **Montager.ijm** provided in **LOBSTER_ROOT/Tools**.

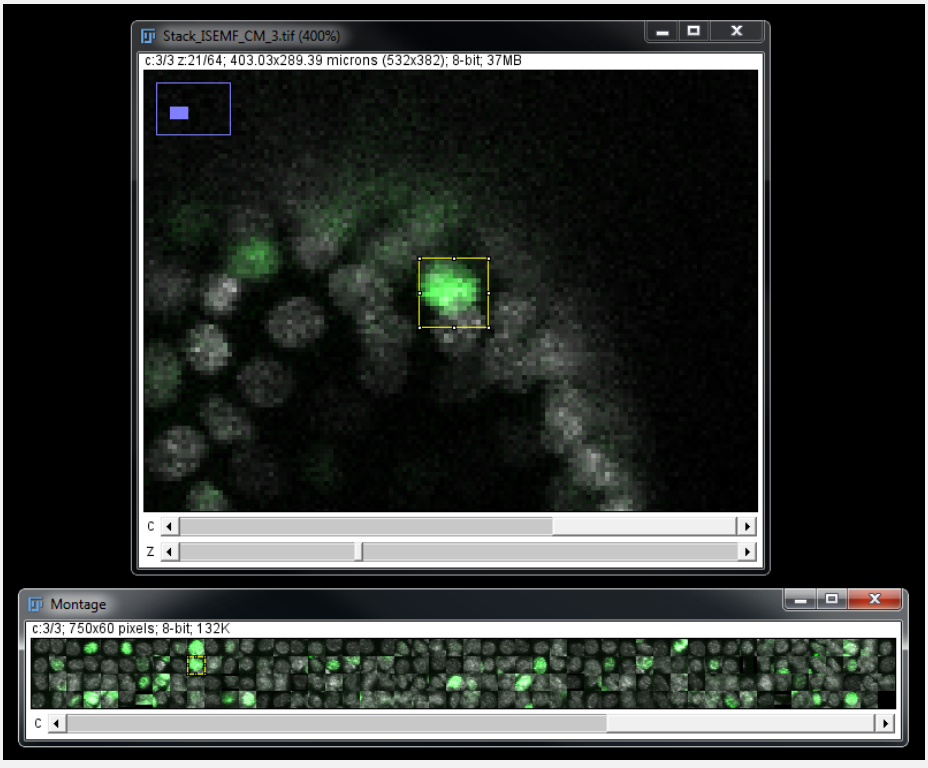
To run the macro, open the original image and drag and drop the corresponding **IRMA** report (‘**Spts**’ or ‘**Objs**’ measurements) to Fiji bar. Also drag and drop the macro to Fiji and launch the macro from the script editor window by pressing “Run”.

This can be tested with the image **LOBSTER_ROOT\Images\TissuePilar3D.tif**

and the report **E:\LOBSTER\Results\Reports\TissuePilar3DNucDet\Stack_ISEMF_CM_3.csv**

obtained after running the journal **TissuePilar3D_NucDet.jls.**

3D boxes are cropped around the detected objects and maximum intensity Z projected before being montaged. To remove an object from the montage, press shift and left click its miniature: it should turn black and a “**1**” is added to the column “**Reject**” at the corresponding row of the results table. If it cannot be concluded if an object should be rejected or not from its miniature, click the miniature to jump to the object location in the input image. To exit the macro cleanly, close the montage.



Montager: Detected targets montage linked to original image

How to use LOBSTER for your own application?

Designing practical journals and becoming familiar with existing **LOBSTER** functions can take some practice but it does not require programming language knowledge: journals are small, human readable scripts configuring functions input/output wiring and parameters, as well as image handling options (display/exportation). To get started, the best approach is probably to test and adapt existing journals; many sample journals with companion images are provided (see Appendix - Applications). To process your own images, you can start by:

1) Look for a journal performing at least part of the analysis you plan; journals are located in **ROOT_LOBSTER/Journals** (see Appendix **applications – journals and job scripts**).
2) Run the journal to get a feeling about the results.
3) If your images are large, you probably first want to crop out some illustrative regions for testing and store them in a folder.
4) Make a copy of the journal and modify it to point to your image folder.
5) Run the journal.
6) You will most probably need to tweak some function parameters to obtain satisfying results: open the journal from the console hyperlink and edit parameters. Save the journal and run it again (**launch** console hyperlink). You can obtain help on a function by typing >> help *FunctionName* in the console or by right clicking on its name and selecting Open "FunctionName" (assuming you edit the journal from MATLAB editor).
7) Completing the project might finally require some customization, such as adding a function to the sequence of operations, changing processing/display options or defining the correct sequence of calls to perform to **JENI**, **IRMA** and **JOSE** to get to the expected results.

Tools to prepare input images (group / Z project / stitch)

Input images (JENI input folder)

Input images should all be stored in the same input folder as 8-bit or 16-bit TIFF files:

- 2D images: channels **must** be separate files (fixed field filenames)
- 3D images: Z slices can be grouped in the same file or stored in subfolders (one per image), channels can be separate files or grouped in same file
- 2D/3D Time-lapses: Z slices **must** be grouped in same file, time frames **must** be separate files and channels stored in **separate** folders.

Images can be Z projected or grouped by time frame by running the ImageJ macro **ImageGroup_or_Project.ijm** located in **LOBSTER/Tools**. This macro is especially useful to convert 3D time-lapses to expected **LOBSTER** format or to Z project "thin" 3D images (see documentation in macro file header). Other macros are also provided to stitch images (XY montage) coming from tiled grid acquisitions (see documentations in macro files). See Appendix **LOBSTER** tools for details.

JOURNALS

Dissecting a 2D image journal (.jl)

LOBSTER applies workflows, sequences of image processing functions to identify objects in input images. **LOBSTER** functions are all stored in **ROOT_LOBSTER/Code/_Functions**.

You do not need to know much about the internals of **LOBSTER** functions but it is important to understand that they have a set of parameters and that they *transform* one or several input images into one or several output images. To get some help on a function: >> help fxg_mWaterTile or open journal in MATLAB editor, right click on a function name and select Open "FunctionName".

Image analysis workflows are defined in **Journals**, small human readable text files configuring input/output folders, wiring between **LOBSTER** functions and function parameters. Journals also configure processing and display options. Sample journals are stored in **ROOT_LOBSTER/Journals**.

Let's inspect the journal file **'Tissue_SegWaterTiles.jl'** from MATLAB editor (Open button from MATLAB upper ribbon, select all Files (*.*) as file filter in file browser):

```
InputFolder = './Images/Tissue/';
OutputFolder = './Results/Images/Tissue/';

@iA = '*.tif';

@fxg_mWaterTiles [iA] > [M];
params.GRad = 4;
params.ExtendedMinThr = 5;
/endf

/show iA > M;
/keep M > tif;
```

Journal 'Tissue_SegWaterTiles.jl'

Input and output folders

The first two lines of a journal are compulsory, they set paths to input and output image folders. Paths can be absolute (e.g. ***E:/LOBSTER/***) or relative to **LOBSTER_ROOT** (starts by **./**).

Input image filter

Images are sequentially processed from input folder: ***@iA*** declares the current input image ***iA*** and associates it to an image filter. Here, the filter configures JENI to exclusively process TIFF files from the input folder. **At least one input image variable should be defined in a journal**.

Note: the character ***** is a wildcard (replaces any string in file name).

Functions

Functions are sequentially called in the order they appear in a journal; they are identified by the special character ***@f*** + FunctionName and arguments. For instance:

```
fxg_mWaterTiles [iA] > [M]
```

calls the function `fxg_mWaterTiles` *transforming* input image ***iA*** into image ***M***

Arguments can be either input image variables or temporary images. Functions parameters are configured in the structure params inside the section *ending by* ***/endf***.

Displaying images

To display input image ***iA*** and overlay (mask) image ***M*****:**

```
/show iA > M
```

Note: To display an image without overlay:

```
/show iA >
```

Exporting images to disk

To export an image to output folder:

```
/keep M > tif
```

Exportation format is specified after '**>**', it is **recommended** to use **tif**.

Batch processing

The same sequence of operations is applied to all input images fulfilling the image filter. Displaying images is useful to test journals but awkward for batch image processing since JENI waits that the user presses '**x**' before resuming processing. To prevent image display and user interaction when launching a journal, use GENI instead of JENI.

Notes:

- Use semicolons to end every lines to avoid outputting information to console
- Interrupt a journal at any time by pressing **Ctrl+C**
- Use **%** to add comments to journal or disable a line without removing it (to disable **/** commands, use **/%show**, **/%keep**)

A *real* workflow: sequencing functions

Typically, several functions are called sequentially in a journal; this is illustrated in the following journal segmenting clusters of nuclei:

```
>> JENI('NucleiCytoo_GradWaterTilesMerge.jl');
```

```
InputFolder = './Images/NucleiCytoo/';
OutputFolder = './Results/Images/NucleiCytoo/';
Lbl = 1;
Fill = 1;

@iA = '*C00*.tif';

@fxg_mGradWaterTiles [iA] > [L];
params.GaussianRadInt = 2;
params.ExtendedMinThr = 1.8;
/endf

@fxm_lTilesMerge [L, iA] > [L2];
params.GaussianRad = 2;
params.MinObjArea = 175;
params.MinSal = -0.5;
params.MaxValleyness = 1.075;
params.GaussianRadGeom = 2;
params.ConcavityThresh = 0.4;
/endf

/show iA > L2;
/keep L2 > tif;
```

Journal 'NucleiCytoo_GradWaterTilesMerge.jl'

Two functions are sequentially called: the first aims at identifying bright regions in the image; the second merges these regions into concave bright particles.

The first function transforms input image **iA** (input folder) into image **L,** input of second function (together with **iA**). The second function outputs image **L2**, exported to file (last line).

To display the intermediary image **L**, add the line `/show iA > L` before the line `/show iA > L2;`

Note: The input image filter is different than before: only TIFF files containing text string '**C00**' are processed; this is required to process a specific channel when several channels are present in input folder.

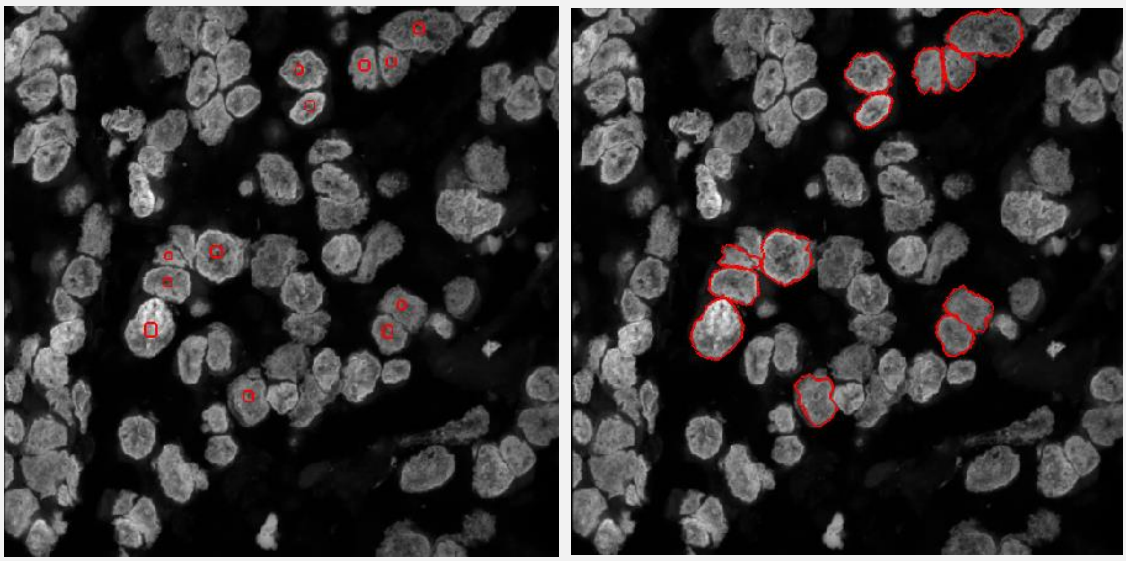
Journals processing several images together

Sometimes, it can be necessary to input several input images to the same function. For instance, the following journal processes input images **'_C00'** together with seed mask images '**_C01**' marking the objects to extract. Two images are then passed to the function `fxgs_lSeededPolarGeod`.

Note: To ensure a correct behaviour, there must be as many input images checking both filters in input folder.

```
InputFolder = './Images/CrazyCells/';
OutputFolder = './Results/Images/CrazyCells/';

@iA = '*_C00*.tif';            % Image filter
@iS = '*_C01*.tif';            % Image seeds filter

@fxgs_lSeededPolarGeod [iA, iS] > [L];
params.ThDiv = 180;
params.ObjRad = 35;
params.MinArea = 175;
params.Method = 'simple';
/endf

/show iA > iS;
/show iA > L;
/keep L > tif;
```

Journal 'CrazyCells_ManualSeededPolarGeod.jl'



Seed image iS overlaid on input image iA (left) image. L overlaid on input image iA (right)

Dissecting a 3D image journal (.jls)

3D journals do not offer as much input/output images flexibility as 2D journals, but they bring much more display options: slice browser, 3D renderer and running Z projection. Importantly, they support 3D bricking, a powerful mechanism to process images of virtually any size.

Let's inspect the journal **'BloodVessels3D_LocThr3DSkl3D.jls'**:

```
% Data section
InputFolder = './Images/BloodVessels3D/';
OutputFolder = './Results/Images/BloodVessels3DSkl/';

% Display/export section
SaveOutput = 1;
ZRatio = 3;
Shw = 3;
RunProj = 4;
PointSize = 5;

% Threshold
params.Sigmas = [2 2 2];
params.MeanBox = [15 15 9];
params.AddThr = 2;
params.IgnoreZero = 0;
M = fxg_mLocThr3D(I, params);

% Skeletonize
params.PreCloseRad = 0;
params.Min2DHolesArea = 50;
params.MinVol = 50;
O = fxm_kSkl3D(M, params);

% Label skeleton
params.SklLbl = 1;
params.MinBrchLgth  = 9;
O = fxk_kSklLbl3D(O, params);
```

Journal 'BloodVessels3D_LocThr3DSkl3D.jls'

3D journals configuration and syntax are a bit different:

- Input and output image variables are always implicitly **I** and **O**
- Only **I** and **O** can be displayed (see variable **Shw** in appendix **journal options**)
- Only **O** can be exported (**SaveOutput** = 1)
- Function definition: [**O1, O2**] **=** FunctionName (**I1, I2**)
- Structure params is defined before function name, and passed as **last** argument.
- For 3D journal, the image **O** is used to compute distance map unless the image **M** is defined. Using both **M** and **O** can be for instance useful to compute skeleton **O** and distance map on non-skeletonized mask **M**.

Handling channels and fixed images in 3D journals

All channels of a 3D image should be stored in the same file; the channel selection is configured from the variables **Step** (default 1) and **Offset** (default 0). For instance, to process every third image skipping first image, set **Step = 3** and **Offset = 1**. Doing so, only these slices are stacked up to form the input image **I**, all other images are ignored.

```
InputFolder = './Images/TissuePilar3D/';
OutputFolder = './Results/Images/TissuePilar3D/';
Step = 3;
Offset = 0;
…
```

Note: Channels are assumed interleaved in 3D image stacks (alternating while browsing the image stacks in natural order).

It is also possible to pass a fixed input image **I2** to a function by defining the variable **FixedInput.** This can for instance be used to pass the same point spread function (PSF) image to the deconvolution function for every input image:

```
InputFolder = './Images/Neurons3D/';
OutputFolder = './Results/Images/Neurons3D/';
FixedInput = './Images/Neurons3D/PSF/PSF_Neurons.tif';
Shw = 0;
SaveOutput = 0;

params.type = 'rl';
…
O = fxg_gDeconv3D(I, I2, params);
```

Intensity measurements over 3D images channels

To perform intensity measurements of multi-channel 3D images:

```
>> [InputFolder MaskFolder] = JENI('TissuePilar3D_NucDet.jls');

>> ReportFolder = IRMA(MaskFolder, '.',  'Objs', [3 3 1], 1, InputFolder);
```

The vector **[3 3 1]** (IRMA 4th argument) reads as '**3D images**', **Step = 3**, **Offset** = 1. That is the channel stack is built by skipping the first image and concatenating every third image of the original stack. To add a second intensity measurement channel with **Offset** = 2, use **[3 3 1 2]**.

Note: When input images are not 3D images but subfolders of 2D images, set **FoldersIn** = 1 in 3D journal and use **'*'** as IRMA 7th input argument (channel filter).

Processing several 3D image channels in a journal

When several channels of a 3D image need to be processed together by a journal, one needs to load the whole 3D image (**Offset** = 0, **Step** = 0) and use the channel splitter function (as many outputs as interleaved channels, up to 3).

```
% Split channels
[I I2 I3] = fx_SplitChans(I);
```

Note: When using channel splitter, make sure that **I** has the same dimensions as the output mask image **O**!

3D Image Bricking

When bricking is enabled, JENI do not load the entire image **I** into main memory, but instead sequentially processes image bricks (cropped out bricks tiling the whole image field).

Each brick has the same number of slices than **I** and a square field of size **Brick + GuardBand** pixels. As many bricks as necessary to tile the entire image are configured. **GuardBand** sets the brick overlap to limit border effects (artificial cuts) between bricks.

When bricking is enabled, for each input image a subfolder is created in output folder to store output bricks; IRMA automatically detects this situation and seamlessly performs the required actions to merge bricks and compute measurements of objects overlapping over several bricks.

Note: '**Trks**' is not compatible with bricking.

Bricked and non-bricked images should never be mixed in the same output folder: it should either contain only TIFF files or subfolders.

Here is an excerpt from a journal configuring bricking:

```
InputFolder = './Images/BloodVessels3D/';
OutputFolder = './Results/Images/BloodVessels3D_brcks/';
SaveOutput = 1;
ZRatio = 3;
Brick = 256;
GuardBand = 64;
Shw = -1;
```

Excerpt from journal 'BloodVessels3D_LocThr3DSkl3D_bricked.jls'

It is highly recommended to use 3D bricking **only** if an image cannot be processed without; this is mostly dependent on the size of the image and the main memory available on the workstation. Brick size should ideally be set to a submultiple of image size (e.g. 512 for 4096 x 4096 images); this optimizes tiling and brings the best computing time performance.

Dissecting a 2D/3D time-lapse journal (.jlm)

In time-lapse journals, functions process all images from input folder together (input/output folders are passed as arguments).

For each call to JENI, only **one time-lapse** is processed (see loopy job scripts to process multiple time-lapses at once).

Input image files can be either 2D or 3D, each time frame **must** be stored in a separate file. Channels **must** be stored in different folders.

The syntax follows 3D journals syntax:

```
% Data section
InputFolder = './Results/Images/HeLaMCF10AMovie/';
OutputFolder = './Results/Images/HeLaMCF10AOvlLbl/';

% Pipeline section
params.SharePenality = 0.33;
params.AlphaCM = 0.33;
params.SmallPart = 15;
params.OvlItExtend = 2;
params.MaxRescueDistance = 48;
fxm_lTrackerOvl(InputFolder,OutputFolder,ReportFolder,params);
```

Journal 'HeLaMCF10A_TrackOvl.jlm'

This example performs nuclei tracking and basic + intensity measurements:

```
>> [InputFolder1 MaskFolder1] = JENI('HeLaMovie_LapThrBinWatTiles.jl');

>> [InputFolder2 MaskFolder2] = JENI('HeLaMCF10A_TrackOvl.jlm');

>> IRMA(MaskFolder2, '.', 'Trks', 2, 1, InputFolder1);
```

Machine learning functions

LOBSTER functions do not implement user interaction, except for three functions: **fxgs_lPatchClassify**, **fxgs_lPatchClassify3D** and **fxg_lBlockClassify**.

These functions classify patches (regions) around seeds or blocks (rectangles) tiling the input images. The classifier is trained from user annotations: a label mask image (function input) or user drawn annotations (user interface). Once classifiers have been trained and saved, these functions behave as *regular* functions without user interaction.

Let's inspect a sample journal calling one of these functions:

```
@iA = '*_C0000*.tif';          % Image filter
@iL = '*_C0001*.tif';          % Image annotations filter

@fxgs_lObjClassify [iA, iS, iL] > [C];
params.FeatType = 'HoG';
params.BoxSize = 7;
params.Sub = 5;
params.NumBins = 8;
params.ClassifierType = 'RF';
params.ClassifierFile = './Classifiers/classifierDABNuc_HoGRF.mat';
params.ExportAnnotations = './Classifiers/Annotation_DABNuc.tif';
/endf
```

Excerpt from journal NucleiDAB_LocMaxHoGRF

The function takes 3 input image arguments: input image, seed image (patch centres) and annotation image (label mask).

If no classifier file is found at **ClassifierFile**, the function first tries to load an annotation file (annotation filter). The user then has the opportunity to edit the annotations or to build them from scratch (follow instructions in slice browser). Finally, a classifier is trained from the annotations and saved to file.

To retrain a classifier, simply erase the corresponding classifier file.

Note: When JENI is launched from JULI, a classifier file **must** be available or an error is triggered: this is because any kind of user interaction is forbidden in this context.

EXPORTING

Exporting objects 3D models from IRMA

Exportation to SWC (filaments network)

Skeleton networks can be exported to *.swc format and imported in external software, such as for instance **Neuromantic** which provides edition, measurements and visualization tools. Exportation to *.swc is performed from IRMA by the following call:

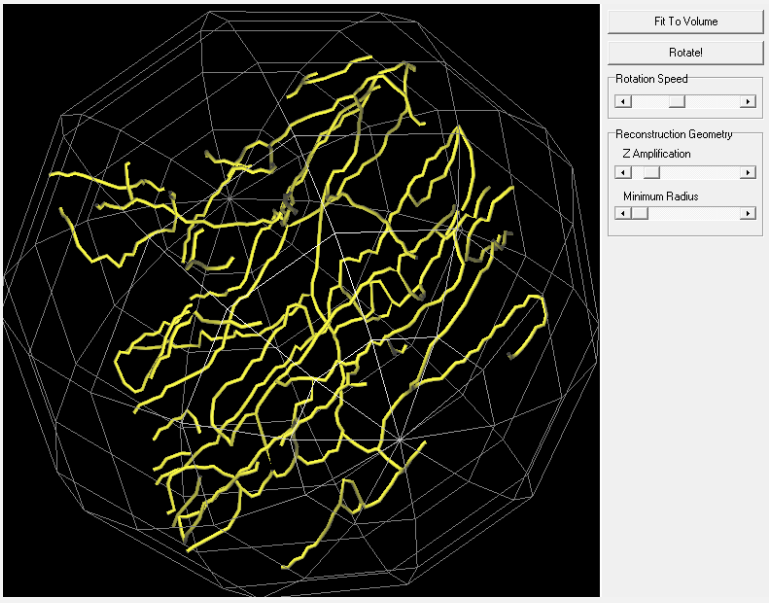
```
[InputFolder MaskFolder] = JENI('BloodVessels3D_LocThr3DSkl3D.jls');

>> ReportFolder = IRMA( MaskFolder, '.', 'Skls', 3 , {3, 4, '.'} );
```

IRMA arguments (SWC format)

5th argument: **{** Z aspect ratio, network tracing step (pix), mesh exportation folder* **}**

* '**.**' → default folder: **LOBSTER_ROOT/Results/Meshes/MaskFolder**

Launch **Neuromantic** (Windows)**.** To open a *.swc file: File > Load Reconstruction. To append a *.swc file: File > Load and Add Reconstruction.



Blood vessel network visualized as filament mesh in Neuromantic

Exportation to STL (object surface mesh)

Objects surfaces can be exported as *.stl, and imported in an external software such as **MeshLab**. Exportation is performed from IRMA by this call:

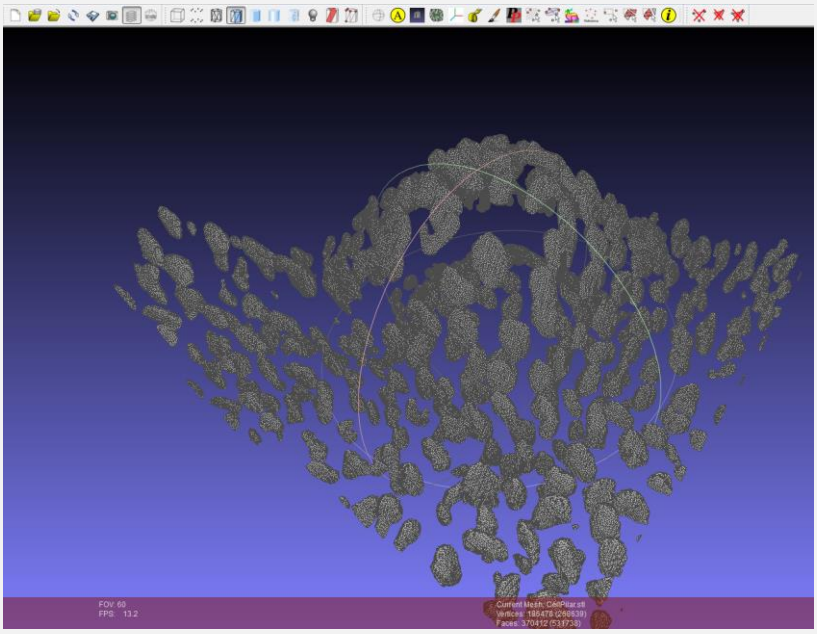
```
[InputFolder MaskFolder] = JENI('CellPilar3D_LogLocMaxLocThrPropagate3D.jls');

>> ReportFolder = IRMA( MaskFolder, '.', 'Objs', 3 , {3, 0.1 , '.'} );
```

IRMA arguments (STL format)

5th argument: **{** Z aspect ratio, mesh sampling, mesh exportation folder* **}**

* '**.**' → default folder: **LOBSTER_ROOT/Results/Meshes/MaskFolder**

Launch **MeshLab.** To open *.stl files: File > Import Mesh…



Objects exported as surface mesh to MeshLab

Exportation to SWC (tracks)

Tracks can be exported to *.swc files and imported in Neuromantic. Exportation is performed from IRMA by this call:

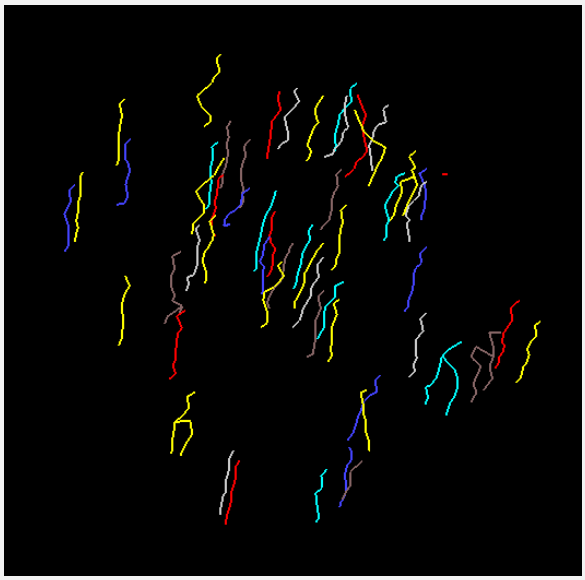
```
>>  [InputFolder MaskFolder] = GENI ('HeLaMCF10A_TrackOvl.jlm');

>> ReportFolder = IRMA( MaskFolder, '.', 'Trks',  2, {1, '', '.'} );
```

IRMA arguments (SWC format)

5th argument: **{** Z aspect ratio (Z time stretch for 2D movies), **''** , mesh exportation folder* **}**

* **'.'** → default folder: **LOBSTER_ROOT/Results/Meshes/MaskFolder**

To open a *.swc file in Neuromantic: File > Load Reconstruction. To append a *.swc file: File > Load and Add Reconstruction. Mode View > Skeletonize Tree is recommended in 3D view.

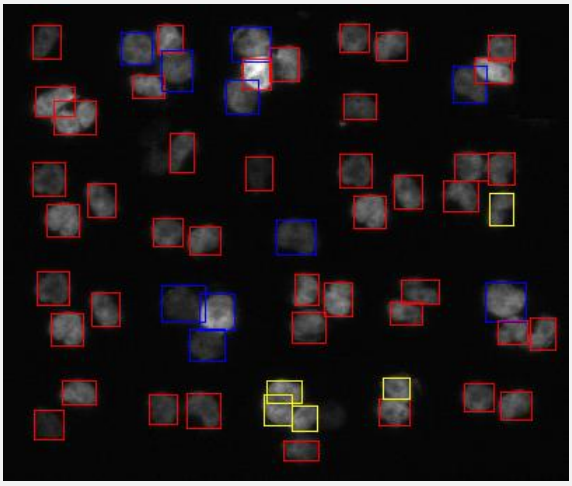


Tracks exported as filament mesh in Neuromantic

JOSE (JOb Scene Exporter)

Exportation of masks and object bounding boxes to ImageJ

Input images, masks and object annotations can be exported to ImageJ as *.sce files by JOSE, and opened by **SceneViewer.ijm** ImageJ macro.



Input image and object bounding boxes (color-coded by object area)

Before using this ImageJ macro, copy the LUT **Random.lut** from **LOSTER_ROOT/Tools** to **Fiji_install_folder/luts**. To run the macro, drag and drop **SceneViewer.ijm** (**LOSTER_ROOT/Tools**) to Fiji bar, and press "**Run"** from the script editor window**.** Select **LOBSTER_ROOT/Results/Reports** as reports folder and pick the scene you wish to open.

Object overlay can be toggled from ROI Manager window ("**Show All**" tick box); channels can be toggled from Channels window.

Some of the jobs from **LOBSTER_ROOT/Jobs** include sample code to export *.sce files; here is an excerpt from **FISH_scene.m**:

```
JOSE(InputFolder1,'*C00*',InputFolder2,'*C01*',ReportFolder1,'Objs',
ReportFolder2,'Spts',ReportFolder1,'IJ', 100, '');
```

JOSE arguments (IJ)

1) Channels: pairs of image channel folder path + image filter
2) Annotations: pairs of report folder paths + object types
3) Folder where to export scene
4) Text string '**IJ**'
5) Channel **Step** + **FoldersIn** flag + **Time-lapse** flag (3 x 0/1 numbers)
6) Optional color-coding, see below

Image filters **must** be text strings starting by '*****'

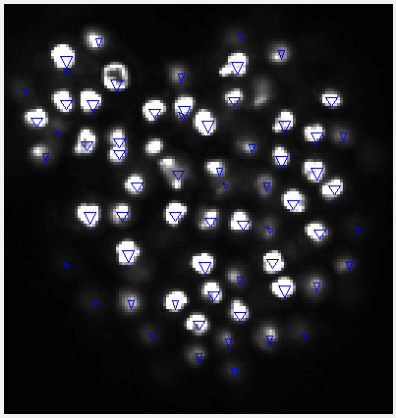
It is also possible to colour-code object bounding boxes (colour 0 to 7) by passing an ImageJ macro line to **JOSE**, for instance:

getResult("Area", **ObjIdx**) >= **250** + getResult("Area", **ObjIdx**) >= **350**

Each item is a Boolean test and can equal 0 or 1 depending on the outcome of the comparison. This expression hence encodes objects with areas **< 250** with default colour (0), object with areas in the range [**250,350**] with colour 1 and objects with areas **>= 350** with colour 2.

Exportation to CellInsight

CellInsight is an ImageJ macro designed to **visualize, edit** and **classify** markers in massive 3D images; it is only compatible with '**Spts**' measurements (spots / objects markers).



Sample screenshot from CellInsight

Here is an excerpt from **LymphGland3D_CellInsight.m**:

```
JOSE(ReportFolder,'Spts', InputFolder,'CellInsight', '', '');
```

The markers are saved as a **.xls** file in same folder as the input image so that they can be easily loaded from **CellInsight** panel. **CellInsight** can be found in **LOBSTER_ROOT/Tools.** Before using it, copy the file **GetString.ijm** to **Fiji_install_folder/Macros**. To launch **CellInsight**, open a 3D image, drag and drop **CellInsight_ vx_x.ijm** to Fiji bar and press "**Run"**. To import a **LOBSTER** scene, press **Import** from CellInsight board. For advance usage, refer to **CellInsight** documentation.

JOSE arguments (CellInsight)

1) Annotations: pairs of report folder paths + object types (must be **'Spts'**)
2) Image **InputFolder** (where *.xls file is saved)
3) Text string '**CellInsight**' followed by two empty strings

ADVANCED FEATURES

Processing several time-lapses or folders: Loopy job scripts

It is possible to batch process several time-lapses by writing a loop over all input subfolders; a template job script is provided below:

```
RootInput = 'E:/LOBSTER/Images/HeLaMCF10AMovie/';
[RootInputFolders, NFolders] = GetFolders(RootInput);
for i=1:NFolders
      CurrentInputFolder = RootInputFolders{i};
      [InputFolder1 MaskFolder1] = GENI('…jl', CurrentInputFolder,-1);
      [InputFolder2 MaskFolder2] = GENI('…jlm', MaskFolder1,-1);
      ReportFolderFolder = IRMA(MaskFolder2,-1,'Trks',2);
end
```

Note the usage of -1 (instead of 1) in the calls to GENI and IRMA: output / report folders are now created 2 levels above current folder (instead of 1 level above).

Note: The same procedure can be applied to process subfolders of input images. This can be useful when images coming from different experiments have same names, as is often the case for high content screening assays (conditions split by folder name).

Checking input images quality

When acquiring many fields of view from an automated microscope, it is often difficult to perfectly control image quality: some images might be out of focus, empty or display artefacts that might be detrimental to automated image analysis.

2D journals can be configured to check the following three conditions and skip images not fulfilling them:

- Focus check is configured by variable **MinLocalFocus** (default to 0). As a rule of thumb image scores below 2-3 are low and can safely be discarded.
- Intensity level check is configured by variable **Min95Percentile** (default to 0): minimum intensity of brightest pixel after discarding 5% brightest pixels.
- Saturation check is configured by variable **MaxSatPixFract** (default to 1): maximum fraction of saturated pixels.

If an image is skipped its result image(s) is/are still exported as black image(s) in output folder.

In case several input images are defined (channels), only the first one is quality checked. For colour images, maximum of RGB channels is computed before performing quality check.

Note: Block size used to compute local score can be configured from variable **LocalFocusBlkSize** (default: 128 x 128 pixels).

Overlap and Co-localization: masks spatial distribution

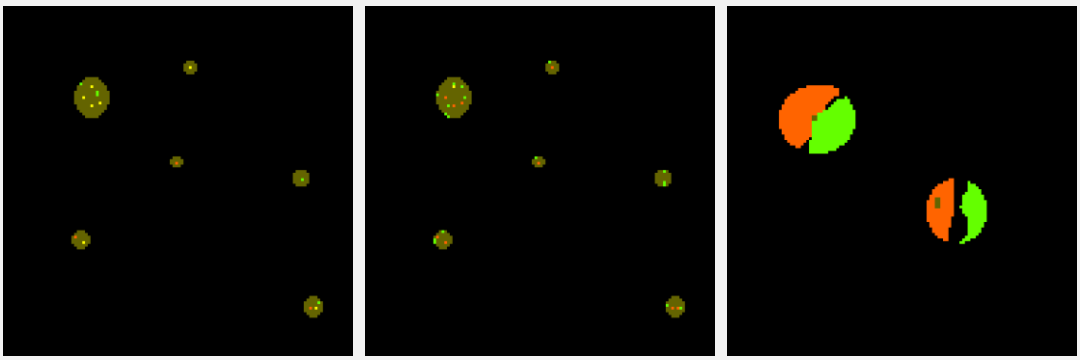
Some more advanced measurements on the spatial (co-)distribution of objects identified in masks can be obtained by calling IRMA 'SpSt' measurement mode.

An example is given in job **LOBSTER_ROOT/Jobs/ProbaColoc.m**

```
NucFolder = './Results/Images/TestColoc/Nuc';
Spt1Folder = './Results/Images/TestColoc/Spt1';
Spt2Folder = './Results/Images/TestColoc/Spt2';

Transfer(NucFolder,Spt1Folder);
Transfer(NucFolder,Spt2Folder);

IRMA(Spt1Folder, '.', 'Spst', 2, {1, 3, ''}, Spt2Folder);
```

ProbaColoc.m job script

For this example, we assume that some objects (e.g. nuclei) have already been segmented by JENI; their binary masks are located in folder **NucFolder.** We also assume that spots were detected in two additional channels; seed masks are located in **Spt1Folder** and **Spt2Folder**.



The three image masks of the example composited

Before calling IRMA, we need to set freedom regions (here the whole nuclei) in seed masks, this is done by calling **Transfer**: a function transferring source object masks to target masks.

The reports for spatial statistics measurements ('**Spst**') include:

- **Name**: Mask name (Channel A, reference mask)
- **Dil**: Dilation radius (used only for **DICE_dil** and FP/FN/TP)
- **Area**: Area of free region, that is, compartments where markers are assumed to move freely and distribute evenly under the “uniform scenario”. It is computed as the number of non-null pixels/voxels in any of the two masks.

Masks overlap analysis (object overlap)

- **Dice**: Dice overlap coefficients (0-1 normalized mask overlap)
- **Dice_ovl**: same as above, but after masks dilation by Dil (default: 3 pix).

Co-localization probabilistic analysis (considering pixels >= 200)

- $\mathbf{F_{rnd}}$: Expected fraction of co-localized pixels for **uniform distribution** in free regions
- $\mathbf{F_{obs}}$: Actual observed fraction of co-localized pixels
- $\mathbf{P_{obs}}$: Probability of the observation assuming **uniform distribution** in free regions

  Low $\mathbf{P_{obs}}$ values (typically <=1%) are evidence of co-localization or exclusion. To conclude on one of these scenarios, one has to compare $\mathbf{F_{obs}}$ to $\mathbf{F_{rnd}}$. For instance, for the 3 pairs of image of this project, $\mathbf{P_{obs}}$ is respectively 1.7 e-8 (very unlikely), 0.34 (likely) and 2.4e-118 (virtually impossible). For the first image pair $\mathbf{F_{obs}} >> \mathbf{F_{rnd}}$, co-localization is hence very likely. For the third image pair $\mathbf{F_{obs}} << \mathbf{F_{rnd}}$ and exclusion is hence very likely. For the second image pair, there is no evidence of co-localization or exclusion.

- **alpha** (co-localization): Probability to observe red positive given green positive normalized to probability to observe red positive given green negative.
- **beta** (exclusion): Probability to observe red negative given green positive normalized to probability to observe red negative given green negative.

For the first pair of images, it is about 34 times (**alpha**) more likely to observe red positive given green positive than given green negative; which clearly hints toward co-localization.

For the third pair of images, it is about 44 times (**beta**) more likely to observe red negative given green positive than given green negative; which clearly hints toward exclusion.

The following measurements can be used to compare object detection in two masks, assuming a spatial tolerance (mismatch) up to (2*__Dil__ + 1) pixels:

- **FN**: False negative: count of objects present in channel A and not in channel B
- **FP**: False positive: count of objects present in channel B and not in channel A
- **TP**: True positive: count of objects present in both channels.

Note: These are just approximations since object assignment is not necessarily 1 to 1.

The following measurements reflect spatial clustering:

- **clustA**: objects A area divided by number of objects A after mask dilation
- **custB**: objects B area divided by number of objects B after mask dilation.

From these measurements, we observe that the third mask pair is heavily clustered (**clustA** and **clustB** around 200) whereas the other two are not (**clustA** and **clustB** around 1).

The default dilation radius can be changed by:

```
>> IRMA(MaskFolder1, '.', 'Spst', 2, {1, rad, ''}, MaskFolder2);
```

Note: The results are exported as a single summary report file 'all.csv' including all images.

JULI (Job UpLInk) : An image analysis server

JULI implements an image analysis server monitoring a user defined folder: when **LOBSTER** job script files (**.m**) are copied to this folder, they are automatically processed and reports sent to a user defined email address upon completion. To launch the server:

>> JULI(*PathToMonitoredFolder*,*PathToLogFile*,'user@gmail.com','password');

JULI arguments

1) Path to folder to monitor
2) Path to error log file
3) Sender email account (optional, no email sent if not defined)
4) Sender account password (optional, no email sent if not defined)

While being processed (by chronological order) job scripts are moved to monitored folder subfolder **Queue**. Upon completion an email is sent with timestamp and notification on the outcome of the job execution. If the variable **AttachmentFolder** is defined, this folder is zipped and attached to email. Email recipient is stored in variable **Dstmail**, if this variable is not defined no email is sent.

If execution fails an error log is emailed and appended to error log file. Depending on the outcome, processed job files are moved to subfolders **Success** or **Error**.

Here is a sample job script for FISH example:

```
InputFolder = 'E:/LOBSTER_sandbox/Images/FISH/';
Dstmail = youremail@youremailprovider.com';

[InputFolder MaskFolder1] = GENI('FISH_nucseg.jl',InputFolder,1);
[InputFolder MaskFolder2] = GENI('FISH_sptdet.jl',InputFolder,2);

AttachmentFolder = IRMA(MaskFolder1,1,'Objs',2,1, MaskFolder2);
```

FISH_sandbox script job

When launched from JULI, a journal never displays any image nor trigger user interaction.

A job script (file name name) can only be processed once per server session, regardless of the outcome of its execution. This is meant to prevent server stall.

JULI is designed to be used together with a local FTP server (e.g. Filezilla): jobs can be committed by FTP uploading files to monitored folder. It is possible to launch several instances of MATLAB running JULI and monitoring the same folder, this is a way to leverage multicore workstations. Some batch scripts are provided for Windows in folder **LOBSTER/Tools** to automatically launch several instances of JULI.

Notes:

MATLAB should have I/O rights to write to journal output / reports folders, as well as monitored folder.

To ensure server security it is recommended to follow the template provided in **LOBSTER_sandox** and provides I/O write to MATLAB **only** inside folders **LOBSTER_ROOT**, **LOBSTER_sandox** and their subfolders (this procedure is OS dependent).

Emailing should work from scratch for Gmail account, other providers might require modifying **LOBSTER** utility function **LOBSTER_ROOT/Code/_Utils/Send_Email**.

Journal re-usability: Relative folder paths

Job scripts are convenient to keep complex workflows but to enhance their re-usability it can be useful to call JENI with two extra arguments: forced input and output folder paths.

A number **X** can actually be passed instead of the output folder: it is then created at same location as input folder (same name + **_oX** appended). The same holds for IRMA **ReportFolder** (appended **_rX**). Here is an example:

```
InputFolder = 'E:/LOBSTER_sandbox/Images/FISH/';
[InputFolder MaskFolder1] = GENI('FISH_nucseg.jl',InputFolder,1);
[InputFolder MaskFolder2] = GENI('FISH_sptdet.jl',InputFolder,2);
IRMA(MaskFolder1,1,'Objs',2,1, MaskFolder2);
```

Excerpt from job script **LOBSTER_sandbox/Jobs/FISH.m**

Now you simply have to update the variable **InputFolder** to process images at different locations. Finally it is possible to override the image channel filter, for instance as:

```
JENI('FISH_sptdet.jl','','',{'*_C01*.tif','*_C02*.tif'});
```

Doing so, every string '*_C01*.tif' in the journal are replaced by a string '*_C02*.tif' prior to execution. **This only works for 2D journals**.

Note 1: **JENI** and **IRMA** clear output folders before each run. As a security mechanism, if the output folder is not empty (and hence needs to be cleared), it must have a parent folder which name holds the string "**Results**" or "**_o**" (can be several levels above).

Note 2: To run job scripts that are not at default locations, open their .m files (double click) and press the green arrow (run) from the editor. In dialog box, press "**Change Folder**".

APPENDIX

LOBSTER Tools

Several tools are available to prepare the images for processing with LOBSTER and explore the results obtained from LOBSTER. They are described in this section and available from **LOBSTER_ROOT/Tools** folder.

Image preparation

Microscopy proprietary formats

When dealing with microscopy proprietary formats, it is advised to use ImageJ to extract the images and export them as multidimensional TIFF files or folders of TIFF images so that they can be processed by LOBSTER.

Z-projecting or grouping images (**ImageGroup_or_Project**.ijm**)**

This ImageJ macro performs Z-projection or grouping of 2D .tif images stored in a folder. The Z-projection / grouping is performed based on a configurable numerical field from the image names. Grouping is useful to prepare images for 3D tracking (one 3D image per time frame). Refer to instructions in the header of the macro (drag and drop to ImageJ to open the macro in text editor).

Montaging / Stitching a mosaic of images (ImageStitcher_ZT_MemHyperStack.ijm)

This ImageJ macro montage / stitch a folder of images from a tiled scan to reconstruct a larger image. The macro can process multidimensional images (Z, T) and save the results as TIFF images. It should only be used if the whole dataset comfortably fit in computer memory (less than 25%). If this is not the case or if you want to process multiple channels, use the following macro instead. Refer to the documentation in the header of the macro (drag and drop to ImageJ to open the macro in text editor).

Montaging / Stitching a mosaic of images (ImageStitcher_ZT_MemHyperStack.ijm)

Alternative macro to previous macro for large datasets.

Results exploration

Montaging identified objects (Montager.ijm)
Refer to Montaging identified objects section.

Overlaying tracked objects (OverlayTrackedObjects.ijm)
Refer to Overlaying tracked objects section to see an application. Refer to the documentation in the header of the macro (drag and drop to ImageJ to open the macro in text editor).

Plotting tracked objects measurements (PlotTrksMeasurements.m)
This MATLAB script opens the results from tracking and plot the measurements for a user specified list of object indices.

LOBSTER Scene viewer (SceneViewer.ijm)
Refer to section **Exportation of masks and object bounding boxes to ImageJ** and documentation in header of the macro (drag and drop to ImageJ to open the macro in text editor).

Appendix Journal results conventions

Object mask (8-bit image):

0	background
255	foreground

Object label mask (16-bit image):

0	background
1-65535	object label

Seed mask (8-bit image):

0	background
100	optional mask (spot segmentation)
200	plain seeds (1 pixel markers)
220-250	classified seeds

Skeleton mask (8-bit image):

0	background
100	optional mask (filament segmentation)
200	skeleton (1 pixel central axis)
220	skeleton end point
250	skeleton branch point

Appendix journal options

Common processing options (2D/3D journals)

Rescale	>=1	Downscaling factor prior to processing (default = 1)
ExportDist	0	Do not export mask distance map (default)
	1	Export distance map (outside objects)
	2	Export distance map (inside objects)
		(Distance map: any pixel >1 is object)
Dilate	>=1	Integer, 2D dilate objects in mask (pix)

2D journals (.jl) specific processing options

MinLocalFocus	Do not process image if focus score smaller than this value
LocalFocusBlkSize	Block size (pix) used to compute local focus score
Min95Percentile	Minimum maximum intensity excluding 5% brightest pixels
MaxSatPixFract	Maximum fraction of saturated pixels

Following options are display only:

Lbl	0	Display binary mask overlay
	1	Display label mask overlay (colours)
Fill	-1	Original mask (no contouring, no dilation)
	0	Mask contour, apply dilation (default)
	1	Original mask + dilation

3D journals (.jls) specific processing options

Step Process only each **Step** images in stack (default = 1)

Offset Start at image **Offset** in stack (default = 0)

SaveOutput If set to 1, output variable **O** is saved to output folder (default = 0)

Brick Brick XY size, defining this variable enables brick mode

GuardBand Overlap in pixels between XY bricks to avoid border effects (default = 64 pix)

FoldersIn Set to 1 if input images are folders of 2D images (default = 0)

Following options affect display only

NCols Sets number of random colours in palette for label masks (default = 16)

PointSize For 3D rendering, sets size of displayed points (default = 1)

RunProj If set >1, running local maximum intensity Z projection is set to **RunProj** slices by default in the stack viewer.

ZRatio For 3D rendering, Z slices to pixel size ratio (default = 1)

Shw Control input and output image stacks display (see below).

- **0** Display input and output images in two separate slice browsers
- **1** Overlay output **binary mask** over input image
- **2** Overlay output binary or label mask contours over input image
- **3** Overlay output label mask over input image (if binary mask detected, connected component analysis is performed to label objects)
- **4** Output volume rendering
- **5** Input volume rendering + output mask overlay as surface points mesh

Time-lapse journals (.jlm) specific processing options

Shw

- -1 No display
- 1 Display label mask overlaid on input image
- 2 Display only output image

Appendix Slice browser and 3D renderer keys / actions

Slice browser

Intensity	Mouse position and image intensity + mask values (lower left corner)
Contrast	Left click + move mouse up/down - left/right to adjust intensity display. Lower and higher clipping values are displayed as **L** (lower) and **W** (higher) and can also be user defined by clicking them.
Histogram	Tick histogram to view image histogram
ROIs	Intensity measurements can be performed by drawing ROIs with tools
Colours	Use drop down menu on the left of floppy disk to change image LUT
Zoom in/out	Press shift + left click + move mouse
Reset view	Use landscape button to reset view
Slice selection	Use slider on left side or mouse wheel to scroll though slices
Grid	Use tick box to display / hide grid
'x'	Resume processing
'r'	Launch 3D renderer (draw bounding box, adjust depth before with **z**)
'z'	Set number of slices used for local z projection
'm'	Toggle mask overlay

3D renderer

m	Show/hide point mesh overlay
c	Display min/max intensity value clipping dialog
space	Reset view
Close window	Resume processing

Appendix IRMA Report measurements

Seeds mask (IRMA third argument: 'Spts')

Centroid	X,Y(,Z) object position
MeanInt	Mean intensity

Skeletons mask (IRMA third argument: 'Skls')

sklvol	Number of pixels (voxels) of the skeleton
skllgth	Total skeleton (min 1 branch) length (XY pixel unit), Z ratio corrected
sklbrpts	Number of skeleton branch points
sklenpts	Number of skeleton end points
objvol	Volume of the segmentation mask prior to skeletonization (pix or vox), if found
imgvol	Image volume (pix or vox)
meanint	Mean intensity of skeleton voxels in intensity channel(s)
histint	Histogram (bins centred on [1:16]) of skeleton voxels in intensity channel(s)

Objects mask (IRMA third argument: 'Objs')

Area	Object area/volume
Centroid	X,Y(,Z) position
Bounding Box	X,Y(,Z) position of top upper left corner + width, height(,depth)
MeanInt	Average intensity inside object in intensity channel(s)
NonNullPix	Number of non-null pixel inside object in intensity channel(s)

Moving Objects mask (IRMA third argument: 'Trks')

Area	Object area/volume at each time point
Centroid	X,Y,(Z) object position at each time point
MeanInt	Average intensity inside object in intensity channel(s) at each time point

Dual mask measurements (IRMA third argument: 'Spst')

Ovl	2nd numeric value of 5th IRMA argument: {ZRatio, Ovl, ''}
Dice	Dice coefficient between both masks
Dice_ovl	Dice coefficient between both masks after dilation by **Ovl** pixels
$\mathbf{F_{rnd}}$	Expected fraction of co-localized pixels (uniform distribution in free regions)
$\mathbf{F_{obs}}$	Observed fraction of co-localized pixels
$\mathbf{P_{obs}}$	Probability of observation for uniform distribution in free regions
alpha	Co-localization probability factor: P(r+|g+) / P(r+|g-)
beta	Exclusion probability factor: P(r-|g+) / P(r-|g-)
FN	False negative: count of objects present in channel A and not in channel B
FP	False positive: count of objects present in channel B and not in channel A
TP	True positive: count of objects present in both channels
clustA	Objects A area divided by number of objects A after mask dilation
custB	Objects B area divided by number of objects B after mask dilation

Notes:

Low P_{obs} (e.g < 1%) hint toward co-localization or exclusion since in both cases the observation will deviate from the free distribution scenario. To conclude on co-localization or exclusion, one should compare $\mathbf{F_{obs}}$ to $\mathbf{P_{obs}}$.

Appendix Applications – journals and job scripts

2D Images - Image Analysis

Nuclei detection	
	NucleiNoisy_DetLogLocMax.jl
Histology	
	NucleiDAB_LocMaxHoGRF.jl
	NucleiDAB_LocMaxRadFeatSVM.jl
	NucleiDAB_LocMaxDeep.jl
Nuclei segmentation	
Clustered nuclei	NucleiCytoo_GradWaterTilesMerge.jl
	NucleiCytoo_GradWaterTilesMerge_checkquality.jl
	NucleiCytoo_BlockOtsuThr.jl
	NucleiCytoo_MultiPassThr.jl
	NucleiCytoo_SBFRegMax.jl
	NucleiCluster_BlobDetPatch.jl
HELA cell nuclei segmentation	HeLa_LapThrBinWatTiles.jl
	HeLaMovie_LapThrBinWatTiles.jl
Cell detection	
Heavily clustered cells	ManyCells_BlobDetRay.jl
Cell segmentation	
From manual seeds	CellsCyto_SeededPropagate.jl
	CrazyCells_1ManualSeededPolarGeod.jl
Automatic	CrazyCells_2BlobDetPatchSeededPolarGeod.jl
	CrazyCells_3LogSeededPolarGeod.jl
Yeasts	Yeasts_BlockOtsuThr.jl
	Yeasts_GradWaterTilesMerge.jl
	Yeasts_MultiPassThr.jl
	Yeasts_RadStdThr.jl
Blood vessels tracing	
Eye	EyeVesselsSpots_ConnLocThrFilamClean.jl
	EyeVesselsSpots_IsoScanFilamClean.jl
Tissue	Vessels_RadStdThrSklMorph.jl
Vesicle/spot detection	
Eye spots mixed with vessels	EyeVesselsSpots_IsoScanSpot.jl
Orientation estimation	
Synthetic Fibers	FibersLocOrientation.jl
Tissue segmentation	
CT	CT_ManualSeededRW.jl

Tissue packing	Tissue_SegWaterTiles.jl
	TissueSimpleMovie_SegWaterTiles.jl
Mitochondria segmentation	
	Mitochondria_RadStdThr.jl
Tissue classification	
Glomeruli identification	Kidney_BlockClassify.jl
Counting objects inside objects	
FISH	FISH_nucseg.m
	FISH_sptdet.m
Objects colocalization	
Synthetic objects	ObjectsColoc_C00.m
	ObjectsColoc_C01.m
Objects relative distance	
Spots to filament distance	VesselsSpots.m
Local orientation	
Fibers	FibersLocOrientation.jl

2D Images - Image pre-processing and mask post-processing

Image enhancement	
Denoising	NucleiNoisy_DenoiseBM3.jl
	NucleiNoisy_DenoiseDCT.jl
	NucleiNoisy_DenoiseNLM.jl
	Zebrafish_Destripe.jl
Phase contrast restoration	CellsPhaseContrast_Restore.jl
Vessels enhancement	EyeFundus_Enhance.jl
Cell junctions enhancement	
	CellJunctions_RestoreFilam.jl
Concave objects splitting	
	RiceGrains_ModBinWat.jl
Points clustering	
	PointClusters_MeanShift.jl
Histology two stain un-mixing	
	TwoPureColors_SplitTwoStains.jl

3D Images - Image Analysis

Spot detection	
Alzheimer plaques	Plaques_LocMaxRadFeatRF3D.jls
Blood vessels	
Tumors / tissues	BloodVessels3D_LocThr3D.jls
	BloodVessels3D_LocThr3DSkl3D.jls
	BloodVessels3D_VessLocThr3DSkl3D.jls
	BloodVessels3D_LocThr3DSklGeod3D.jls
	BloodVessels3D_LocThr3D_diam.jls
	Brain3D_LocThr3DSkl3D.jls
Fungi	
	Fungi3D_LocThr3DSkl3D.jls
Apical cell segmentation	
	EmbryoTissue3D_LocThrFitSurfFindCells3D.jls
Nuclei detection	
	CellPilar3D_DetLog3DLocMax3D.jls
	LymphGland3D_DetLog3DLocMax3D.jls
	Nuclei3D_DetLog3DLocMax3D.jls
	Nuclei3D_TubesToFilaments3DLocMax3D.jls
	TissuePilar3D_NucDet.jls
Nuclei segmentation	
	CellPilar3D_LogLocMaxLocThrPropagate3D.jls
3D surface segmentation	
	TissuePilar3D_LocThrFitSurf3D.jls

3D Images - Image pre-processing and mask post-processing

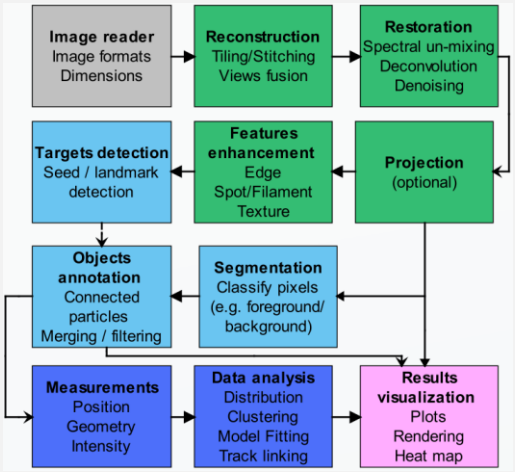
Deconvolution	
	Neurons3D_Deconv3D.jls
Tubes enhancement	
	BloodVessels3D_TubesToFilaments3D.jls
Vessels enhancement	
	BloodVessels3D_Vesselness
Stack focuser	
	CellColonies3D_StackFocuser3D.jls
Seeds clustering	
	PointClusters3D_MeanShift3D.jls

Time-lapse image analysis

Spot tracking (2D)	
Test time-lapse	FakeTrackMovie_TrackDst.jlm
Nuclei tracking (2D)	
Dividing HeLa cells nuclei time-lapse	HeLaMCF10A_TrackOvl.jlm
Junction delimited object tracking (2D)	
Drosophila epithelium cadherin time-lapse	TissueSimpleMovie_TrackMemb.jlm
Object tracking (3D)	
Test time-lapse	TestMovie3D_TrackOvl3D.jlm
Object tracking (3D)	
Test time-lapse	TestMovie3D_TrackDst3D.jlm
Particles velocimetry (2D)	
Von Kármán swirling flow time-lapse	PIV.jlm
Events detection	
Dividing nuclei time-lapse	DividingCells_LocMaxDeep3D.jls

Appendix - Functions

BioImage (BIA) workflows are sequences of functions, usually implementing algorithms from the categories represented in the following figure.



A typical BIA workflow

LOBSTER functions are located in folder **Code/_Functions**, their names are built as:

fx + input/output image type indicators + "_" + explicit name + **3D** (if 3D processing)

G	grayscale
C	RGB color image
M	object binary mask
L	object label mask
S	seed mask
K	skeleton mask

For instance, the function *fxgs_lObjClassify* performs object classification, takes a 2D grayscale image and a 2D seed mask input and outputs a 2D label mask. 2D functions can also be used to process 3D images stored as folders of 2D images.

The best way to know about a function is probably to type help + function name in MATLAB console. There you'll get a description of the function, its arguments and parameters. Importantly, a hyperlink link to a representative journal using the function is provided in the help. As an entry point, follows a quick overview of the functions provided in **LOBSTER**.

Image reader and reconstruction

Image access is handled by **JENI**, image reconstruction has to be performed externally from the ImageJ macros provided in **LOBSTER_ROOT/Tools**.

Image restoration

Advanced denoising and deconvolution functions are available: *fxg_gDeconv3D*, *fxg_gDenoiseBM3*, *fxg_gDenoiseDCT*, *fxg_gDenoiseNLM*, *fxg_gDestripe* (lightsheet microscopy). More conventional denoising (Gaussian blur) is available in many functions as adjustable sigma pre-processing step. The function *fxc_cggSplitTwoColorStains* performs spectral un-mixing on 2-stain color images (histology) and *fxg_gRestorePhaseContrast* restores objects as bright on a dark background in phase contrast microscopy images. Finally, *fxg_gIllumCorr* can be used to compensate field illumination.

Intensity Z-projection

Stack projection can be performed by *fxg_gStackFocuser3D* for brightfield images. Conventional maximum/mean Z intensity projections are not implemented since **LOBSTER** can process images in their natural dimensions, such projections can however be performed externally from the ImageJ macros provided in **LOBSTER_ROOT/Tools**.

Features enhancement

Features enhancement is provided for 3D tubes/filaments (*fxg_gTubesToFilaments3D, fxg_gEnhanceVessels3D and fxg_gEnhanceVessels2D*) and membranes / filaments (*fxg_gRestoreFilam*). Spot enhancement is not a standalone function but embedded in spot detection functions and complex images showing discriminative textures can be classified by machine learning functions (see below).

Target/objects detection

Spot detection is implemented in *fxg_sBlobDetPatch*, *fxg_sBlobDetRay* and *fxg_mIsoScanSpot*. Alternatively, spots or object seeds can be detected as intensity local maxima by *fxg_sLocMax(3D)* and *fxg_sLoGLocMax(3D)*. Finally, strong central symmetry points can be identified by *fxg_sSBFRegMax*.

Segmentation

Generic segmentation can be achieved by *fxg_mMultiPassThr* and *fxg_mBlockOtsuThr*. Small irregular objects or spots can be segmented by *fxg_mRadStdThr* and blob like objects by *fxg_mLapThrBinWatTiles*. Local thresholding is implemented in *fxg_mLocThr3D*. For more complex problems, an image can be decomposed in small blocks classified by *fxg_lBlockClassify(3D)*.

As a robust segmentation technique, *fxg_mWaterTiles* computes superpixels for membrane staining (hollow objects). *fxg_mGradWaterTiles* performs the same operation but applies edge detection pre-filter (plain objects). The function *fxm_lTilesMerge* attempts to merge superpixels as bright convex objects. Filaments can be segmented by generic techniques or by *fxg_kConnLocThrFilam* and *fxg_kIsoScanFilam*.

Objects annotation

Convex particles of binary masks can be splitted apart by *fxm_mModBinWat*, skeleton gaps and small branches fixed by *fxkg_kSklClean*. Once segmented, filaments centerlines can be identified by *fxm_kSklMorph, fxm_kSkl3D* and *fxm_kSklGeod3D (analysed skeleton output)*. The function *fxk_kSklLbl3D* analyses skeleton (branch/end points), **IRMA** can also do it but will not save analysed images. Object tracking can be performed by *fxm_lTrackerOvl(3D)* and small objects / spot tracking by *fxm_lTrackerDst(3D)*. Once detected, seeds and their surroundings can be classified by *fxgs_lPatchClassify(3D)*, and surrounding objects segmented by *fxgs_lSeededPropagate(3D)*, *fxgs_lSeededRW* (Random walker) and *fxgs_lSeededPolarGeod* (minimum geodesic contour).

Special functions

fxg_gmLocOrientation computes local image orientation, *fxg_gmPIV* computes particle velocimetry from a sequence of images and *fxm_mFitSurf3D* fits a 3D surface to an open layer. Seed markers can be clustered by *fxs_lMeanShift* (2D/3D) and *fxsm_sSeedTissue3D* identifies closed hollow objects distributed on a 3D surface. Some generic functions are provided to split 3D stack channels (*fx_SplitChans*), filter connected particles (2D/3D) by size (*fxm_mFilterObjSize*), discard 2D/3D connected particles touching image edges (*fxm_mClearBorders*), only keep pixels with a given value in a 2D/3D image (*fxm_mSieveValue*) and mark centroids of 2D/3D connected particles (*fxm_sMarkObjCentroids*).

Results visualization

Results visualization is performed while running JENI (image and stack viewer, 3D renderer) or by exporting scenes with **JOSE**.

Adding your own functions

To add your own function:

1) Use **LOBSTER** function naming convention
2) Output image should have **same size** as input image,
3) Output images should be 8 or 16 bit and comply to mask formats convention
4) Add extra last input argument **params** structure to pass functional parameters
5) Output arguments must be separated by comas
6) No image display or user interaction should be included in the function
7) Document function and ideally add sample journal hyperlink (see example)
8) Copy function to **LOBSTER_ROOT/Code/_Functions**

```
function [It] = fxg_mLocThr3D(I, params)

        % 3D local threshold.
        %
        % Sample journal:
        % <a href="matlab:JENI('CellPilar3D_LocThr3D.jls');">CellPilar3D_LocThr3D.jls</a>
        %
        % Input: Image stack
        % Output: Binary mask stack
        %
        % Parameters:
        % Sigmas:       Gaussian blur X,Y,Z radii (vector)
        % MeanBox:      X, Y, Z box size to compute local mean (vector, pix)
        % AddThr:       Minimum difference to local mean
        % IgnoreZero:   If set to 1 ignore zeros in local mean computation

        %% Your Matlab code goes here

end
```