



**The *bip* software
(Biological Image Processing)**

User Manual (wip)

Philippe Andrey et al.

June 2021–July 2025

Contents

1	Introduction	4
1.1	Image types	4
1.2	Numerical types	4
1.3	Batch processing	5
1.4	Pipelines	5
1.5	Image file formats	6
2	Installation instructions	7
2.1	Dependencies	7
2.2	Linux installation: Ubuntu 22.04	7
2.2.1	Installing the dependencies	7
2.2.2	Completing the installation	8
2.3	Linux installation: Ubuntu 24.04	8
2.4	Windows installation	8
3	Basic operations	10
3.1	Image properties	10
3.2	Image compression	11
3.3	Image structure	12
3.4	Image arithmetics	13
4	Image transforms	15
5	Filters	19
6	Segmentation operators	23
6.1	Intensity thresholding operators	23
6.2	Watershed transform operators	24
6.3	Label operators	25
6.4	Segmentation-related operators	26
6.5	Evaluation of segmentation results	26
7	Mathematical morphology	28
7.1	Binary operators	28
7.2	Grayscale operators	31
8	Analysis operators	37
8.1	Label operators	37
8.2	Distance maps	41
8.3	Measurements	43
8.4	Export operators	44
9	Pipelines	47
9.1	Writing and using pipeline files	47
9.2	Pipelines without pipeline files	48
9.3	Pipeline variables	48

10 Advanced usage	50
10.1 Neighbourhood systems and connectivity	50
10.2 Pattern substitutions on file and directory names	50
11 Global options	53
12 Project-specific operators	55
13 BIP in action: illustrated examples	57
13.1 Black tophat	57
13.2 Selecting objects based on size	57
13.3 Selecting objects based on size and shape	58
13.4 Labelling clusters of neighbouring objects	61
13.5 Cell distance maps	62

List of Figures

1	Cropping an image from a label image	15
2	Masking a gray level image from a segmented image	17
3	Estimating image derivatives using the Gaussian gradient operator . . .	20
5	Highlighting under- and over-segmentation errors	27
6	Operators for binary mathematical morphology: inner and outer rings. .	29
8	Selection of labels and of their immediate neighbours.	39
12	Separating touching objects	42
15	Matching processed image files with additional files (simple situation) .	51
16	Matching processed image files with additional files (typical situation) .	52
17	Selecting objects based on size	59
18	Selecting objects based on size and shape	60
19	Labelling clusters of neighbouring objects	62
20	Computing cell distance from tissue center	63

1 Introduction

BIP is a command-line interface tool for processing and analysing images, with a specific bias towards microscopy images that are typically generated in developmental and cell biology research studies. The software integrates many standard algorithms as well as specific algorithms we have developed in our own research projects. Many functionalities are quite generic image processing operators, while others address specific needs typically encountered in bioimage analysis. We initially developed BIP for our own needs and now make it available to a wider audience in the hope it will be useful to others.

1.1 Image types

The first major feature of BIP is to support all images types and numerical types that are typically encountered in bioimage processing. This includes all combinations of image dimensions (2D and 3D), number of channels (such as different fluorescent channels), number of time-points (as acquired in time-lapse experiments). Vectorial images are also supported, in which two or more values can be assigned to each image position. Such images covers true color RGB images (as typically processed in histological studies) but also gradient images (with one gradient component per image dimension), results of Fourier transforms etc. Though such images could be represented as multi-channel images, there are some theoretical and practical motivations for having this specific representation. Among others, this also implies that multi-channel vectorial images are also supported.

1.2 Numerical types

The second major feature is that BIP supports all standard numerical types, from the most usual ones in microscopy (unsigned 8-bits and unsigned 16-bits) to less classical ones. The complete list include unsigned char (8-bits), signed char (8-bits), unsigned short (16-bits), signed short (16-bits), unsigned int, signed int, float (known as 32-bits in ImageJ/Fiji) and double ([Table 1](#)). This allows in particular to implement pipelines in which the desired numerical precision is preserved throughout. A specific operator implements conversions between all numerical types (see **convert** operator below).

BIP operators automatically detect input image type and numerical type from the input images, thus offering a user-transparent support for all possible numerical types. Almost all operators support all types and write the output image in the same numerical type as the input image. Some operators store the output image in a different type. This is the case for example for operators computing real values on output (e.g., Euclidean distance transform or Fourier transform).

Type	Bits	Min	Max	Note
uint8	8	0	255	A.k.a. 8 bits (Fiji)
int8	8	-128	127	
uint16	16	0	65535	A.k.a. 16 bits (Fiji)
int16	16	-32768	32767	
uint32	32	0	4,294,967,295	
int32	32	-2,147,483,648	2,147,483,647	
float	32	-3.4e+38	3.4e+38	A.k.a. 32 bits (Fiji)
double	64	-1.8e+304	1.8e+304	

Table 1: Numerical types supported in BIP. The types are listed by their names as used in BIP operators (such as the **convert** operator). For each type, the table gives the number of bits used to store a value, and the minimum and maximum values that can be represented. Equivalence with the nomenclature used in ImageJ/Fiji (when available) is given in the last column.

1.3 Batch processing

The third major feature of BIP is to be designed for batch processing of image datasets. BIP can take as input one or several images and process them according to the specified operator. As a command-line tool, BIP can rely on the features of the shell to process complete set of images or only subsets defined using specific filename patterns. For example, to apply an operator to all images within an **input** directory, simply enter:

```
shell$ bip <operator> ../input/*.tif
```

For example, provided a standardized filename nomenclature (such as using the ISO 8601 standard to represent dates) has been adopted (which is more than highly recommended), the images acquired in October, November and December 2017 and 2019 on DAPI-stained samples would be typically processed by the following command:

```
shell$ bip <operator> ../input/*201[79]-1[012]-*DAPI*.tif
```

1.4 Pipelines

The fourth major feature of BIP is to allow the implementation of pipelines. Pipeline syntax has been designed to be as simple as possible. As a result, a pipeline is simply defined by the list of the corresponding operators and obey the same syntax as the one used when invoking each operator in turn. The sequence of operators composing a pipeline is typically listed in a text file. However, an option is also available to define pipeline upon invocation on the command line. See [Section 9](#) to learn more on the advantages of pipelines and how to use them.

1.5 Image file formats

One limitation of BIP is the absence of support for a large number of image file formats. Indeed, the main supported format for input images in BIP is the TIFF file format and its BigTIFF declination. However, tools are available (for example in ImageJ/Fiji) for converting images from almost any format to TIFF. In addition, BIP also supports some formats derived from TIFF, such as Zeiss LSM image file format.

By default, BIP stores output images as compressed TIF files, using the lossless LZW compression scheme. This may be an issue for some image readers. Compression as a default writing mode can therefore be disabled using the global `-u` option (see [Section 11](#)). In addition, users can use the BIP **uncompress** operator to uncompress their files at times where they process them with specific readers, before using the **compress** operator for long-term storage when done.

2 Installation instructions

2.1 Dependencies

BIP depends on a number of specific external libraries for running:

- LibTIFF - TIFF Library and Utilities (<https://libtiff.gitlab.io/libtiff/>)
- FFTW - Fastest Fourier Transform in the West (<https://fftw.org/>)
- HDF5 - High-performance data management and storage suite (<https://www.hdfgroup.org/solutions/hdf5/>)
- yaml-cpp - A YAML parser and emitter in C++ (<https://github.com/jbeder/yaml-cpp>)

2.2 Linux installation: Ubuntu 22.04

2.2.1 Installing the dependencies

You need super-user rights to install the required dependencies. We assume here you are under the Ubuntu linux distribution, and that you are listed in the sudo-ers.

Install LibTIFF:

```
shell$ sudo apt install libtiff5
```

Install FFTW:

```
shell$ sudo apt install libfftw3-dev
```

Install HDF5:

```
shell$ sudo apt install libhdf5-103-1 libhdf5-cpp-103-1
```

Install yaml-cpp:

```
shell$ sudo apt install libyaml-cpp0.7
```

2.2.2 Completing the installation

Download the distributed BIP executable from the website. Save the file in a directory, typically `$HOME/bin`. We recommend to use a directory that is included in your `$PATH` environment variable. This way, you can access BIP from any location on your system without having to type the full access path to the executable file.

The `$HOME/bin` directory is generally already listed in the `$PATH` environment variable. If not, you can add the following lines to your `$HOME/.profile` hidden file:

```
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

Check that you have executable permissions on the BIP file. This can be done from the file navigator, by right-clicking on the file and then going into the Permissions tab, clicking "Allow execution" (under Ubuntu). Alternatively, this can be done by opening a terminal, going into the directory where BIP is installed, and entering the command:

```
shell$ chmod a+x bip
```

To check that the installation is correct, open a terminal and simply enter:

```
shell$ bip
```

The command should display BIP usage and the list of available operators.

2.3 Linux installation: Ubuntu 24.04

The procedure is the same as above, except you have to replace `libtiff5` by `libtiff6`, and `libyaml-cpp0.7` by `libyaml-cpp0.8`.

2.4 Windows installation

Download the distributed archive for Windows from BIP website. Extract the folder contained in this archive to the desired location on your system, typically in Program Files. Add the extracted folder to your `Path` environment variable.

To check that the installation is correct, open a command-line interpreter (`cmd.exe`) or command-line shell (`PowerShell` or `MSYS2` shell) and enter:

```
shell$ bip
```

The command should display BIP usage and the list of available operators.

3 Basic operations

3.1 Image properties

`convert [-f] <uint8 | uint16 | uint32 | int8 | int16 | int32 | float32 | float64>`

Converts the input image to the specified numerical type (see [Table 1](#) for the characteristics of the different numerical types available in BIP).

Conversion may cause truncation when converting between types of different ranges, such as from `uint16` (16-bit image) to `uint8` (8-bit image). Conversion may also cause a loss of precision, as when converting from `float32` to `int32`. By default, the operator raises an error if the conversion would result in truncation or loss of precision. This behaviour can be overridden using the `-f` option, which forces the conversion to be performed even if such errors occur.

`info [-l] [-o <output-file[.tsv]>]`

This operator prints a table of properties for the input images. The properties include image size, number of channels and timepoints, numerical type, spatial calibration, and compression mode. See [Table 2](#) for a complete listing of reported parameters.

This operator is particularly useful to check that images have a correct spatial calibration, and that a collection of images in a project have consistent spatial calibrations or numerical types.

If the `-l` option is specified, the table displays in addition the minimum and maximum value in each image.

The output of this operator may look messy and difficult to read in the terminal, especially when applied to large sets of images with filenames of varying size (which breaks the alignment of column contents). You can use the `-o` option to specify an output filename (or the shell redirection operator) to store the output in a file that can subsequently be loaded in your preferred spreadsheet application. The format of the output file is Tabular-Separated-Values. The `.tsv` extension is automatically added if absent from the specified argument of the `-o` option.

`set dx[,dy,dz,dt,x0,y0,z0] | unit <value>`

Warning: since this operator modifies metadata, its use is strongly discouraged, unless you know exactly what you are doing. This operator is provided mainly for repair operations, for example to re-introduce in metadata the spatial calibration that may have been lost during processing with 3rd-party software (yes, this happens).

This operator sets metadata values related to spatial calibration, temporal calibration, and position in physical space. For example, the spacing between consecutive slices in

Column	Information
file	Input image filename (including access path, if any)
type	Numerical type of image values (see supported types in Table 1)
compression	Compression mode (none or lzw)
sizeX	Number of columns in the image
sizeY	Number of rows in the image
sizeZ	Number of slices in the image (equals 1 for 2D images)
samples	Number of samples (values) per pixel/voxel
timepoints	Number of timepoints in the image
channels	Number of channels in the image
dx	Spatial sampling in the horizontal direction (“pixel width”)
dy	Spatial sampling in the vertical direction (“pixel height”)
dz	Spatial sampling between slices (“voxel depth”)
unit	Length unit
dt	Temporal calibration (time interval between frames in time-lapse)
x0	Position of top-left corner in physical space (X coordinate)
y0	Position of top-left corner in physical space (Y coordinate)
z0	Position of top-left corner in physical space (Z coordinate)
minimum	The minimum value contained in the image [optional]
maximum	The maximum value contained in the image [optional]

Table 2: Image parameters listed by the **info** operator.

a 3D image would be set using:

```
shell$ bip set dz 0.707 *.tif
```

Several values can be set at once, as shown in this example for the XY sampling values:

```
shell$ bip set dx,dy 0.618 *.tif
```

The length unit is specified by its scale (power of 10, in meters). For example, to specify that lengths are expressed in microns, one should use:

```
shell$ bip set unit -6 *.tif
```

3.2 Image compression

compress

This operator compresses the input image files (lossless compression). The compression algorithm is the Lempel-Ziv-Welch (LZW) method ([Welch, 1984](#)), which is also described in the TIFF specifications ([Adobe Developers Association, 1992](#)). No output file is written if the input file is already compressed.

uncompress

This operator uncompresses the input images. No output file is written if the input file is already uncompressed. This operator may be useful to allow or to speed-up file loading in software in which the opening of compressed image files is not supported or is too slow.

3.3 Image structure

merge-channels

This operator takes as input a list of images and merges them as different channels into a multi-channel image as output. The input images must have the same XYZ size, the same numerical type, the same number of samples per pixel, and the same number of timepoints.

Note the input images can be themselves multichannel images. The total number of channels in the output image is thus the sum of the number of channels in the input images. This feature allows for example to add a new channel to a pre-existing multi-channel image.

merge-slices <dz>

This operator takes as input a list of 2D image files and concatenates them into a 3D image stack as output. The input 2D images must have the same XY size, the same numerical type, the same number of samples per pixel, and the same numbers of channels and of timepoints.

The parameter **dz** is the Z-spacing between the slices in the generated stack (a.k.a. voxel depth). It should be expressed in the same length unit as the XY calibration of the input images.

For example, one would call the following to merge slices with a 0.123 Z-spacing:

```
shell$ bip merge-slices 0.123 slice1.tif ... sliceN.tif
```

merge-timepoints

This operator merges input images along their temporal dimension. Any number of images, containing each an arbitrary number of timepoints, can be merged. The only condition that the images must satisfy is that their other dimensions (XYZC), their value types, and their spatial calibrations are identical.

The output filename is generated from the basename of the last specified image.

split-channels

This operator splits the different channels contained in the input image into as many separate files. Individual channel files are numbered starting from 0. Filenames are padded with '0's to ensure a constant filename size.

No output file is generated if the input image contains a single channel.

split-slices

This operator splits the input 3D image into as many separate files corresponding to its 2D slices. Individual slice files are numbered starting from 0. Filenames are padded with '0's to ensure a constant filename size.

The generated 2D images contain the same number of channels and timepoints as the input image.

No output file is generated if the input image is not a 3D image.

split-timepoints

This operator splits the different timepoints contained in the input image into as many separate files. Individual timepoint files are numbered starting from 0. Filenames are padded with '0's to ensure a constant filename size.

No output file is generated if the input image contains a single timepoint.

3.4 Image arithmetics

abs

This operator replaces the values of the input image by their absolute values.

add <image>

This operator performs the point-to-point addition with the specified **image**.

The operation is only possible between images of same size and same numerical type.

add-value <value>

This operator adds the specified **value** to the input image.

divide <image>

This operator performs the point-to-point division with the specified **image**.

The operation is only possible between images of same size and same numerical type.

divide-value <value>

This operator divides the input image with the specified **value**.

multiply <image>

This operator performs the point-to-point addition with the specified **image**.

The operation is only possible between images of same size and same numerical type.

multiply-value <value>

This operator multiplies the input image by the specified **value**.

subtract <image>

This operator performs the point-to-point subtraction with the specified **image**.

The operation is only possible between images of same size and same numerical type.

subtract-value <value>

This operator subtracts the specified **value** to this image.

4 Image transforms

crop-from [-m <crop-margin>] <label-image-path>

This operator computes the bounding boxes of the labelled objects found in the label image specified by **label-image-path** and uses these bounding boxes to crop the input image(s) into as many sub-images (Figure 1). The output sub-images are stored in separate files, with filenames obtained by appending label numbers to the basename of the input image file.

The **-m** option can be used to add a margin around the crop region. The margin width is expressed in pixels/voxels.

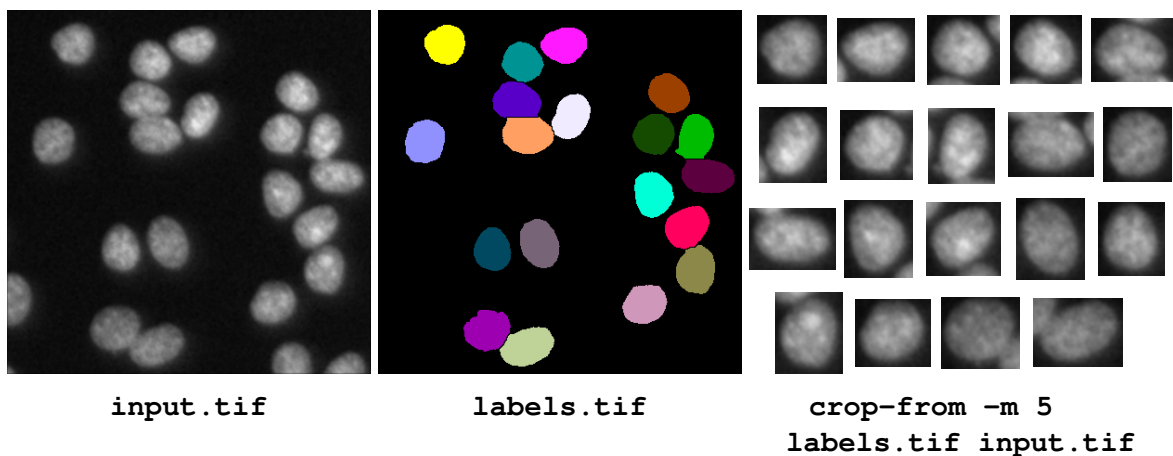


Figure 1: Cropping an image from a label image: operator **crop-from**.

The **crop-from** operator can also be used to split the segmentation masks of the different objects present in a segmented image. In this case, the label image is specified twice on the command line, first as the image to compute the crop boxes, and second as the image to be cropped:

```
shell$ bip crop-from -m 1 labels.tif labels.tif
```

fft

This operator computes the fast Fourier Transform of the input image. Only images with 1 sample per image position can be processed (multi-channel images are supported). The output images have 2 samples, the first one storing the real part and the second one storing the imaginary part of the complex numbers of the Fourier Transform. Note that the origin (zero-frequency component) is not centered in the output image (no swap of quadrants).

flip x|y|z

This operator flips the contents of the input image along the specified direction (symmetry transform). Specifying 'x' corresponds to a left/right symmetry. Specifying 'y'

corresponds to a top/bottom symmetry. Specifying 'z' corresponds to a stack top/s-tack bottom symmetry in a 3D image. As expected, using the 'z' option on a 2D image does not modify its contents.

invert auto | type | <value>

This operator inverts the values of the input image. The inversion is performed according to the following formula, applied at every image position:

$$\text{invertedValue} \leftarrow \text{pivotValue} - \text{originalValue}$$

The pivot value depends on the parameter passed to the operator:

auto the pivot value is taken as the maximum value in the image;

type the pivot value is taken as the maximum possible value for the numerical type of the image (255 for unsigned char, 65535 for unsigned int, etc.);

<value> the pivot value is set to the specified `value`.

isoscale nearest | linear

This operator resizes the input image using interpolation in order to obtain an output image with an isotropic spatial calibration (i.e., as if its spatial sampling was isotropic). For example, an input image with voxel size of 0.5,0.5,1.0 micron would have its number of slices doubled by this operator, thus yielding a voxel size of 0.5,0.5,0.5 micron.

The operator automatically determines which direction(s) should be resampled. The direction(s) with the smallest spatial sampling is preserved, and the other direction(s) are upsampled. Note that because of the discrete nature of digital images, it is not always possible to obtain an exactly isotropic element size.

The parameter of the operator specifies the interpolation mode. As a rule of thumb, it is recommended to use the **linear** mode in all cases except when processing binary or label images, for which averaging values is meaningless. For these images, the parameter should be set to **nearest** (no interpolation).

mask <mask-image-path>

This operator performs a masking operation by setting to 0 all positions of the input image that fall out of the specified mask. A position is considered as located within the mask if its value in the mask image differs from 0. Note this allows to use mask images that are not necessarily binary images (such as label images; see [Figure 2](#)), though many practical applications will probably use this operator with binary images.

The mask image may have a different size (hence, spatial sampling) than the image to filter. In this case, nearest interpolation neighbour is used to map positions between the two images. This is typically useful when defining a mask at a coarse resolution and applying it to another image at a higher resolution.

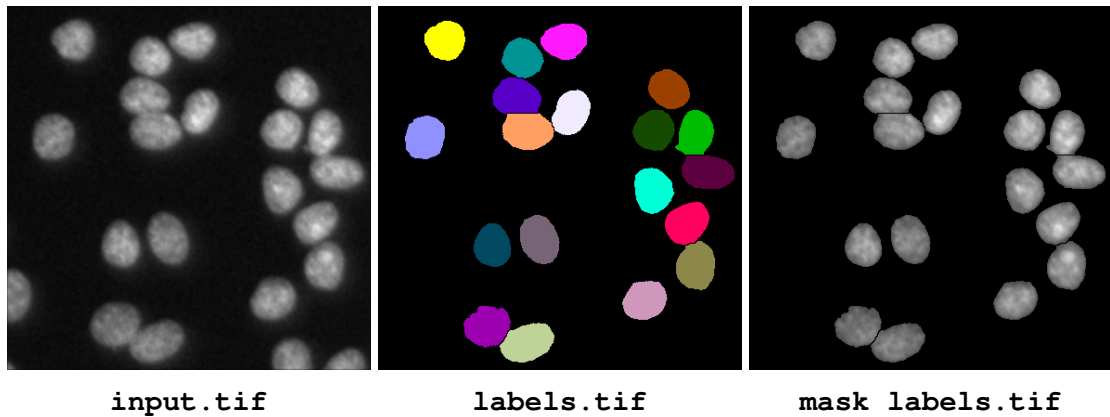


Figure 2: Masking a gray level image from a segmented image: operator **mask**. Complete command to generate the output image: **bip mask labels.tif input.tif**.

The mask image and the image to filter can also have different value types. This is useful since mask images are typically 8-bit (when binary), while images to filter can be of any type.

norm

This operator computes the norm of the input image. This operator is meaningful for vector images only, i.e., images in which several values (samples in the TIFF nomenclature) are associated to each image position (such as gradient or FFT images). The computed norm is the Euclidean (L2) norm:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=0}^{n-1} x_i^2}$$

projection avg | max | min

This operator performs the 2D projection of the input image along the Z direction. The input image is assumed to be 3D. In the 2D case, the output is identical to the input.

The parameter of the operator specifies the projection mode. If set to **avg**, values are averaged along the Z direction for each XY position. If set to **max**, the maximum value along the Z direction is retained for each XY position (this also known as MIP, Maximum Intensity Projection). And yes, **min** does what you expect it does.

reslice x | rx | y | ry

This operator reslices the input image along the specified axis and direction. The image is assumed to be 3D (the image is left unchanged by the operator otherwise). The reslicing is controlled by the parameter passed to the operator:

- x** reslices the image from left to right;
- rx** reslices the image from right to left;
- y** reslices the image from top to bottom;

ry reslices the image from bottom to top.

scale nearest | linear <factor>

This operator resizes the input image in all directions according to the specified scale `factor`. The spatial calibration of the image is updated accordingly.

The first parameter of the operator specifies the interpolation mode. As a rule of thumb, it is recommended to use the `linear` mode in all cases except when processing binary or label images, for which averaging values is meaningless. For these images, the parameter should be set to `nearest` (no interpolation).

resample nearest | linear <dx,dy[,dz]>

This operator resizes the input image using interpolation so that the resulting image has the specified `<dx, dy [, dz]>` spatial calibration. When processing 2D images only, `dz` can be omitted. Be warned, however, that `dz` defaults to 1.0, so that the voxel depth of 3D images can be modified even if this argument is not specified.

Note that because of the discrete nature of digital images, it is not always possible to obtain an output image with exactly the same spatial calibration as the one specified by the user.

The first parameter of the operator specifies the interpolation mode. As a rule of thumb, it is recommended to use the **linear** mode in all cases except when processing binary or label images, for which averaging values is meaningless. For these images, the parameter should be set to **nearest** (no interpolation).

5 Filters

`attenuation-correction [-p] ball | box <radius> <reference-slice>`

This operator filters the input 3D image so as to correct the attenuation of signals with depth, using Biot et al's method (Biot et al., 2008). The method relies on the hypothesis that the background of the image should be stationary. Hence, a background image is estimated for each slice and a correction is computed to bring its background average and standard-deviation to that of a reference slice. The background of each slice is obtained by applying a morphological opening. The method is thus appropriate to correct attenuation in image stacks containing relatively small objects such as spots.

The first two parameters of this operator control the morphological opening. See the **opening** operator for details. The third parameter **reference-slice** is the index of the slice taken as reference for the correction. It is generally taken as one of the first slices in the image stack.

If the **-p** option is set, the per-slice average intensity profiles of the original image, of its background estimated based on the morphological opening, and of the corrected image are stored in a dataframe file. This can be used to plot and visualize the strength of the attenuation and the quality of the correction.

`gaussian-filter <sigma>`

This operator filters the input image by replacing each value by a weighted average of values over its neighbourhood, the weights being set according to a Gaussian function. This is a classical linear filter that is in general used to attenuate noise.

The **sigma** parameter controls the shape of the Gaussian function. Though the neighbourhood is theoretically infinite, it is truncated at ± 3 **sigma**, where weights are becoming negligible relatively to the central position.

`gaussian-gradient <sigma>`

This operator computes the Gaussian gradient of the input image. The principle of this operation is to apply a Gaussian smoothing followed by a differentiation along each direction (the way it is implemented differs from this description for the sake of efficiency). For a N -dimensional image ($N = 2$ or 3), the result is a vector image with N values (samples in the TIFF nomenclature) corresponding to the N components of the gradient (Figure 3).

The **sigma** parameter controls the shape of the Gaussian function used at the smoothing stage (see **gaussian-filter** operator).

The Gaussian gradient is an approximate optimal edge detection filter (Canny, 1986).

This operator cannot be applied to multi-sample input images.

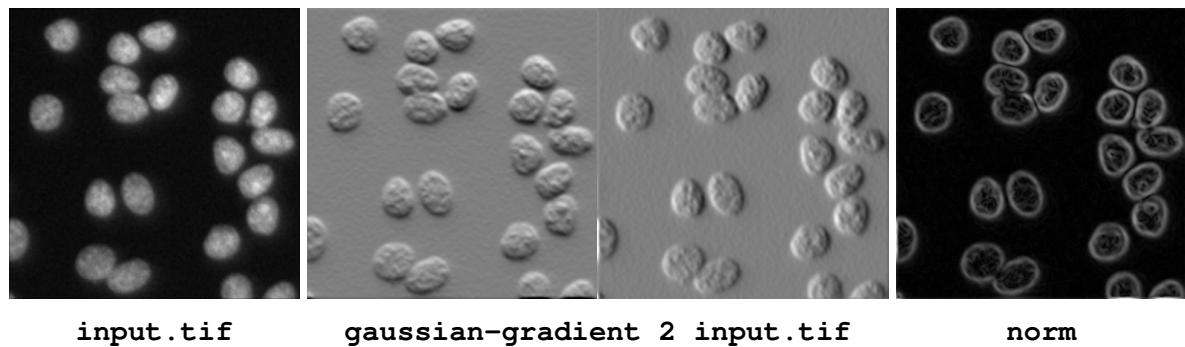


Figure 3: Estimating image derivatives using the Gaussian gradient operator. The operator generates a vector image of the gradient components (*Middle*). The gradient norm is obtained by applying the **norm** operator to the vector image (*Right*).

local-inhibition-filter ball | box <radius>

This operator filters the input image by first setting to zero all values that are below some proportion (scale parameter) of the maximal value of their neighbourhood. In a second step, the obtained image is used as a marker for a geodesic reconstruction of the input image, thus recovering original values for positions associated to significant intensity peaks.

This operator is useful to remove intensity peaks that are considered not relevant in the neighbourhood of relevant (and of higher intensity) peaks. It was initially developed to remove artefacts in 3D-SIM images (Keller et al., 2024).

The first parameter of this filter specifies the shape of the neighbourhood. When set to **ball**, circular (2D) or spherical (3D) neighbourhoods are used. When set to **box**, square (2D) or cubic (3D) neighbourhoods are used. The size of the neighbourhood is controlled by the **radius** parameter.

The scale parameter is fixed to 0.2 (may become an option in the future).

majority-filter ball | box <radius>

This operator filters the input image by replacing each value by the most represented value in its neighbourhood. It is primarily intended to filter label images by removing irregularities at the boundaries of labeled regions. Therefore, the value 0 is ignored. An option may be added in the future to alter this behavior.

The first parameter of this filter specifies the shape of the neighbourhood. When set to **ball**, circular (2D) or spherical (3D) neighbourhoods are used. When set to **box**, square (2D) or cubic (3D) neighbourhoods are used. The size of the neighbourhood is controlled by the **radius** parameter.

mean-filter ball | box <radius>

This operator filters the input image by replacing each value by the average value in its

neighbourhood. This is a basic linear filter that is used in particular to attenuate noise.

The first parameter of this filter specifies the shape of the neighbourhood. When set to **ball**, circular (2D) or spherical (3D) neighbourhoods are used. When set to **box**, square (2D) or cubic (3D) neighbourhoods are used. The size of the neighbourhood is controlled by the **radius** parameter.

median-filter ball | box <radius>

This operator filters the input image by replacing each value by the median value in its neighbourhood. This is a basic non-linear filter that is used in particular to attenuate noise. When applied to a binary image, it is equivalent to a local majority voting.

The first parameter of this filter specifies the shape of the neighbourhood. When set to **ball**, circular (2D) or spherical (3D) neighbourhoods are used. When set to **box**, square (2D) or cubic (3D) neighbourhoods are used. The size of the neighbourhood is controlled by the **radius** parameter.

morphological-gradient ball | box <radius>

This operator computes the morphological gradient of the input image. The morphological gradient is the difference between the dilation (local maximum filter) and the erosion (local minimum filter) of the image. One advantage of this operator over linear gradient operators is to be less sensitive to variations in the local curvature of object contours.

The first parameter of this operator specifies the shape of the neighbourhood (structuring element) used in the dilation and erosion operations. When set to **ball**, circular (2D) or spherical (3D) neighbourhoods are used. When set to **box**, square (2D) or cubic (3D) neighbourhoods are used. The size of the neighbourhood is controlled by the **radius** parameter.

plantseg [--config <config-file>] [--model <model-name>]

This operator runs a Convolutional Neural Network called PlantSeg (Wolny et al., 2020) to enhance boundary signals in input images. It is typically applied before a watershed transform on images of labelled cell membranes or walls to segment cells within tissues.

PlantSeg is actually a 3D-Unet architecture (Falk et al., 2019) that has been pre-trained on plant tissue images. Users should take care of having PlantSeg installed on their system.

The default model has been trained on 3D confocal image stacks of *Arabidopsis thaliana* ovules. A different model can be specified using the **--model** option.

prewitt-gradient

This operator computes the Prewitt gradient of the input image. The Prewitt operator is a separable linear gradient operator with smoothing kernel 1 1 1 and derivative kernel -1 0 1. In 2D, for example, the resulting kernel to compute the horizontal component of the gradient is thus:

$$\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}$$

The Prewitt gradient is generally of little interest on biological images. It is essentially provided for historical and pedagogical reasons, and may be useful in very specific applications.

sobel-gradient

This operator computes the Sobel gradient of the input image. The Sobel operator is a separable linear gradient operator with smoothing kernel 1 2 1 and derivative kernel -1 0 1. In 2D, for example, the resulting kernel to compute the horizontal component of the gradient is thus:

$$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

The Sobel gradient is generally of little interest on biological images. It is essentially provided for historical and pedagogical reasons, and may be useful in very specific applications.

unsharp-mask-filter <alpha> <sigma>

This operator filters the input image by performing an unsharp mask operation. This operation consists in adding to the image some proportion of the difference between the image and a smoothed version of it. The net effect is to enhance the local contrast by making transitions at the boundary of objects sharper.

The parameter **alpha** controls the proportion of difference added to the original image. Only positive values should be passed. The larger the value, the larger the contrast enhancement. The parameter **sigma** is passed to the Gaussian filter used to smooth the image. Its value should be adapted to the width of the transitions to be enhanced (i.e., to the degree of blur in the input image).

variance-filter ball | box <radius>

This operator filters the input image by replacing each value by the variance of the values in its neighbourhood. It may be used for example to enhance positions where important intensity variations occur or can be used as a basic texture analysis tool.

The first parameter of this filter specifies the shape of the neighbourhood. When set to **ball**, circular (2D) or spherical (3D) neighbourhoods are used. When set to **box**, square (2D) or cubic (3D) neighbourhoods are used. The size of the neighbourhood is controlled by the **radius** parameter.

6 Segmentation operators

6.1 Intensity thresholding operators

threshold <threshold-value>

This operator performs a binarization of the input image by simple thresholding. Values that are strictly below the specified **threshold-value** in the input image are set to 0 in the output image. Input values that are equal or above the threshold are set to 1.

otsu-thresholding

This operator performs a binarization of the input image by thresholding using a threshold value automatically computed following Otsu's method (Otsu, 1979). The criterion for determining the best threshold is the separability between the two classes defined by the threshold. In Otsu's method, separability is formally defined as the ratio between the between-class variance and the within-class variance. Intuitively, this corresponds to selecting a threshold that optimises simultaneously the contrast between the two classes and the homogeneity within each class.

isodata-thresholding

This operator performs a binarization of the input image by thresholding using a threshold value automatically computed following the method of Ridler and Calvard (Ridler and Calvard, 1978). The method iterates separating the histogram into two classes using the current threshold value, computing the class averages, and setting the threshold for the next iteration as the middle point between the two averages. This is a 1D version of the k -means algorithm (with $k = 2$), also known as Isodata algorithm. This method is also related to Otsu's method (Xue and Zhang, 2012), and generally gives, at best, similar results.

unimodal-thresholding

This operator performs a binarization of the input image using a threshold value computed according to the "triangle" method (Zack et al., 1977), also known as unimodal thresholding (Rosin, 2001). As the name indicates, this method was designed to compute intensity thresholds from histograms that do not exhibit strong bimodality, as is the case when the objects of interest cover a small portion only of the image. Our implementation assumes objects are bright over a dark background. In the reverse situation, the image should be inverted before computing the threshold. For example:

```
shell$ bip pipeline -e "invert auto | unimodal-thresholding"
```

6.2 Watershed transform operators

watershed [-n 4,6 | 8,26]

This operator runs the classical watershed transform: starting from the regional minima of the input image, labels are progressively growing as if (virtually) flooding the image until they meet other labels.

The connectivity used for computing regional minima and for propagation in the watershed algorithm is set using the `-n` option. See [Section 10.1](#) for more information about the meaning and usage of this option.

To obtain relevant segmentation results with this operator, the input image should have high intensity values at the interfaces between objects. Applying a gradient operator is required if this is not the case, i.e., if the input image contains labelled objects rather than labelled object boundaries.

h-watershed [-n 4,6 | 8,26] <h>

This operator runs the classical watershed transform after having filtered the input image. The filtering operation aims at reducing the over-segmentation problem that is frequently encountered when applying the classical watershed transform alone. The filtering consists in removing from the image the minima that are not significant, i.e., whose depth is below the specified `h`-value.



Input image (16 bits)

`bip watershed`

`bip h-watershed 5000`

The connectivity used at the filtering step and for propagation in the watershed algorithm is set using the `-n` option. See [Section 10.1](#) for more information about the meaning and usage of this option.

To obtain relevant segmentation results with this operator, the input image should have high intensity values at the interfaces between objects. Applying a gradient operator is required if this is not the case, i.e., if the input image contains labelled objects rather than labelled object boundaries.

This operator is provided for convenience. It is equivalent to the following pipeline (using $h = 10$ in this example):

```

# converts image to 16-bit unsigned integer values
# (no effect if image is already 16-bit unsigned)
# and stores a copy for later retrieval in the pipeline
convert unsigned-short
store $image

# removes non-significant minima
extended-minima 10
store $emin
recall $image
minima-imposition $emin

# applies watershed on the filtered image
watershed -n 8,26

```

marker-watershed [-n 4,6 | 8,26] <marker-image>

This operator runs the watershed transform using user-specified seed for initialization. The seeds, or marker, are used to pre-filter the input image and to remove regional minima not corresponding to any seed.

The seeds are taken from `marker-image`. Any connected set of non-zero values in this image is considered as a seed.

The connectivity used to define connected sets in the marker image and for propagation in the watershed algorithm is set using the `-n` option. See [Section 10.1](#) for more information about the meaning and usage of this option.

To obtain relevant segmentation results with this operator, the input image should have high intensity values at the interfaces between objects. Applying a gradient operator is required if this is not the case, i.e., if the input image contains labelled objects rather than labelled object boundaries.

6.3 Label operators

labelling [-n 4,6 | 8,26]

This operator performs a labelling of the connected components in an image. A connected component is a set of image positions within which any two positions can be connected by a path that never passes by background positions. The background is defined as the positions with value 0. These definitions of components and background imply in particular that this operator can be applied to binary images as well as to non-binary images. In the latter case, connected components are defined as sets of connected non-null values.

The connectivity used to define connected sets is set using the `-n` option. See [Sec-](#)

tion 10.1 for more information about the meaning and usage of this option.

condense-labels

This operator performs a renumbering of the labels in an image in order to obtain a consecutive sequence of label numbers. This is useful for example when some labels have been removed or merged. Applying this operator ensures in particular that the maximal value in a label image equals the number of labeled objects.

Note that a call to the `labelling` operator is generally not equivalent to a call to the `condense-labels` operator, as the first one would merge labels that are connected. In addition, the second one runs faster than the first one, since it operates on label values only and not on their spatial structure.

It is likely that users will use this operator almost exclusively on integer-type images. If needed (for example to avoid type conversion within a pipeline), it should be noted that it can as well be applied to real-type images, as long as the actual values are integers.

BIP makes no distinction between intensity and label images (this distinction is the responsibility of the user). Hence, although this operator is primarily intended for application on label images, it can also be applied to intensity images (though it is unclear which situation would benefit from such processing).

6.4 Segmentation-related operators

isosurface [--stl] <threshold-value>

This operator computes the isosurface at the specified value in a 3D image. The isosurface is defined as the set of positions that take the specified value and is computed using linear interpolation between voxel positions. The implementation follows a corrected version of the Marching Cubes algorithm (Lorenson and Cline, 1987).

The computed isosurface is represented as a triangular mesh. By default, the output file format is the `svviewer` shape file format (`.tm` file).

Use the `--stl` option to store the computed isosurface in the STL format (e.g., for 3D printing).

6.5 Evaluation of segmentation results

label-errors split | merge | split-merge <reference-image>

This operator selects the labels corresponding to under- and over-segmentation errors

in the input image compared to the specified reference image (Figure 5). The errors are detected by first computing a forward mapping giving for each label of the reference image the label that intersects most in the input image. Similarly, a backward mapping is computed between the input image and the reference. An under-segmentation error occurs when two or more labels of the reference image are mapped to the same label in the input image. Reciprocally, an over-segmentation error occurs when two or more labels of the input image are mapped to the same label in the reference image.

The first argument of the operator is used to select the type of error that is computed. If set to **split-merge**, both under- and over-segmentation errors are computed. In this case, the output image contains two channels. The first channel gives the labels of the reference image that are under-segmented in the input image. The second channel gives the labels of the input image that are splitting some labels in the reference image.

If the first argument is set to **merge** or **split**, only the corresponding type of error is computed and the output image contains a single channel.

Restrictions: this operator cannot process multi-channel or multi-sample images.

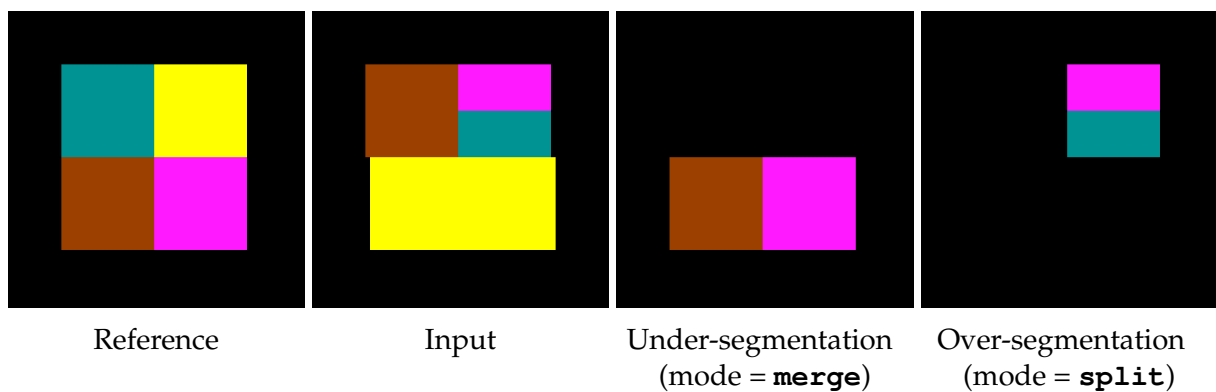


Figure 5: Highlighting under- and over-segmentation errors with **label-errors**.

7 Mathematical morphology

7.1 Binary operators

binary-closing ball | box <radius>

This operator performs a binary morphological closing on the input image. The image is supposed to be binary, with the background set to 0 and the foreground set to 1. This operator provides the same result than the more general closing operator (see **closing**) but its implementation specific to binary images is more efficient.

The first parameter of this filter specifies the shape of the structuring element (neighbourhood). When set to **ball**, circular (2D) or spherical (3D) structuring elements are used. When set to **box**, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the **radius** parameter.

binary-dilation ball | box <radius>

This operator performs a binary morphological dilation on the input image. The image is supposed to be binary, with the background set to 0 and the foreground set to 1. This operator provides the same result than the more general dilation operator (see **max-filter**) but its implementation specific to binary images is more efficient.

The first parameter of this filter specifies the shape of the structuring element (neighbourhood). When set to **ball**, circular (2D) or spherical (3D) structuring elements are used. When set to **box**, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the **radius** parameter.

binary-erosion ball | box <radius>

This operator performs a binary morphological erosion on the input image. The image is supposed to be binary, with the background set to 0 and the foreground set to 1. This operator provides the same result than the more general erosion operator (see **min-filter**) but its implementation specific to binary images is more efficient.

The first parameter of this filter specifies the shape of the structuring element (neighbourhood). When set to **ball**, circular (2D) or spherical (3D) structuring elements are used. When set to **box**, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the **radius** parameter.

binary-inner-ring <width>

This operator computes the inner ring of objects in a binary image. The **width** parameter controls the size of the ring. The inner ring is defined as the part of objects that is removed during their erosion (Figure 6):

$$\text{InnerRing}(I) = I - (I \ominus B_w)$$

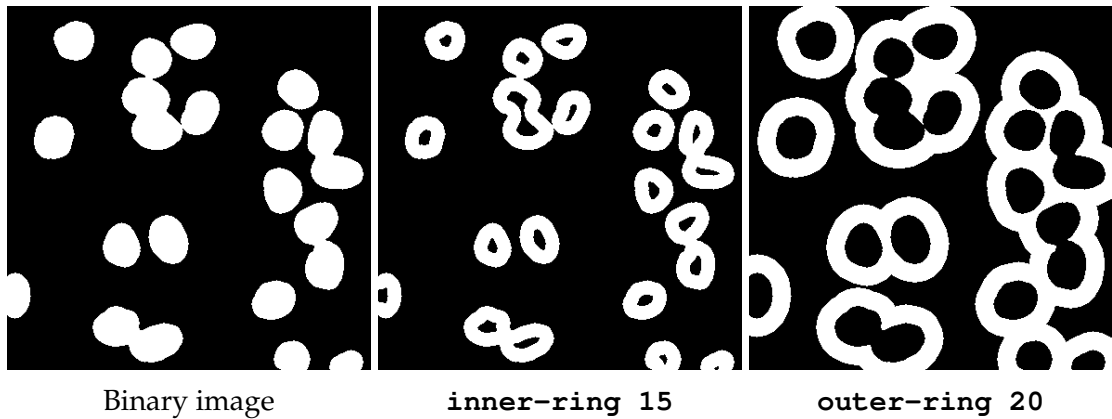


Figure 6: Operators for binary mathematical morphology: inner and outer rings.

where $I \ominus B_w$ denotes the erosion of image I by a circular (2D) or spherical (3D) structuring element B_w of radius w . Note that the inner ring includes the object contours.

binary-opening ball | box <radius>

This operator performs a binary morphological opening on the input image. The image is supposed to be binary, with the background set to 0 and the foreground set to 1. This operator provides the same result than the more general opening operator (see **opening**) but its implementation specific to binary images is more efficient.

The first parameter of this filter specifies the shape of the structuring element (neighbourhood). When set to **ball**, circular (2D) or spherical (3D) structuring elements are used. When set to **box**, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the **radius** parameter.

binary-outer-ring <width>

This operator computes the outer ring of objects in a binary image. The **width** parameter controls the size of the ring. The outer ring is defined as the part of objects that is added during their dilation (**Figure 6**):

$$\text{OuterRing}(I) = (I \oplus B_w) - I$$

where $I \oplus B_w$ is the dilation of image I by a circular (2D) or spherical (3D) structuring element B_w of radius w . Note that the outer ring does not include object contours.

fast-binary-closing <radius>

This operator performs a binary closing on the input image. The structuring element is a disk in 2D and a sphere in 3D. The output is the same as the one obtained with the **binary-closing** operator using a ball as structuring element. However, the **fast-binary-closing** operator runs in constant time (with regards to the size of the structuring element). It is thus more efficient when using large structuring elements.

The **radius** parameter controls the size of the structuring element. It is expressed in

pixels (2D) or voxels (3D).

fast-binary-dilation <radius>

This operator performs a binary dilation on the input image. The structuring element is a disk in 2D and a sphere in 3D. The output is the same as the one obtained with the `binary-dilation` operator using a ball as structuring element. However, the `fast-binary-dilation` operator runs in constant time (with regards to the size of the structuring element). It is thus more efficient when using large structuring elements.

The `radius` parameter controls the size of the structuring element. It is expressed in pixels (2D) or voxels (3D).

fast-binary-erosion <radius>

This operator performs a binary erosion on the input image. The structuring element is a disk in 2D and a sphere in 3D. The output is the same as the one obtained with the `binary-erosion` operator using a ball as structuring element. However, the `fast-binary-erosion` operator runs in constant time (with regards to the size of the structuring element). It is thus more efficient when using large structuring elements.

The `radius` parameter controls the size of the structuring element. It is expressed in pixels (2D) or voxels (3D).

fast-binary-opening <radius>

This operator performs a binary opening on the input image. The structuring element is a disk in 2D and a sphere in 3D. The output is the same as the one obtained with the `binary-opening` operator using a ball as structuring element. However, the `fast-binary-opening` operator runs in constant time (with regards to the size of the structuring element). It is thus more efficient when using large structuring elements.

The `radius` parameter controls the size of the structuring element. It is expressed in pixels (2D) or voxels (3D).

fill-holes [-n 4,6 | 8,26]

This operator fills holes present in a binary image. A hole is a set of background values completely surrounded by foreground values. Completely surrounded means these background values are not connected to a connected set of background values that reach the image border.

The connectivity used to define the neighbourhood system when computing the connected components is set using the `-n` option. See [Section 10.1](#) for more information about the meaning and usage of this option.

ultimate-erosion [-n 4,6 | 8,26]

This operator computes the ultimate eroded set of the input image. When progressively eroding a binary image, the ultimate eroded set is defined as the union of the connected component that disappear at each erosion step. In practice, the ultimate eroded is computed as the regional maxima of the distance transform of the input image (Soille, 2003). In our implementation, the output image contains the values of the distance transform in the ultimate eroded set.

The connectivity used to define the neighbourhood system when computing the connected components is set using the `-n` option. See Section 10.1 for more information about the meaning and usage of this option.

7.2 Grayscale operators

asf-closing-opening ball | box <first-radius> <last-radius>

This filter smooths the input image through a sequence of alternated closing and opening operations, starting first with a closing. It is useful to smooth out noise or other undesired structures when they cover a range of different scales and introduces less distortion than applying opening and closing with a large structuring element (Sternberg, 1986).

The first parameter of this filter specifies the shape of the neighbourhood (structuring element) used in openings and closings. When set to **ball**, circular (2D) or spherical (3D) neighbourhoods are used. When set to **box**, square (2D) or cubic (3D) neighbourhoods are used.

The radius of the neighbourhood is varied step-by-step from **first-radius** to **last-radius**. Hence, running this operator with a ball neighbourhood and with **first-radius** set to 1 and **last-radius** set to 3 is equivalent to applying the following pipeline:

```
closing ball 1
opening ball 1
closing ball 2
opening ball 2
closing ball 3
opening ball 3
```

asf-opening-closing ball | box <first-radius> <last-radius>

This filter smooths the input image through a sequence of alternated opening and closing operations, starting first with an opening. It is useful to smooth out noise or other undesired structures when they cover a range of different scales and introduces less distortion than applying opening and closing with a large structuring element (Sternberg, 1986).

The first parameter of this filter specifies the shape of the neighbourhood (structuring element) used in openings and closings. When set to **ball**, circular (2D) or spherical (3D) neighbourhoods are used. When set to **box**, square (2D) or cubic (3D) neighbourhoods are used. The radius of the neighbourhood is varied step-by-step from **first-radius** to **last-radius**.

closing ball | box <radius>

This operator performs a morphological closing on the input image. Closing is a combination of a dilation (see **max-filter**) followed by an erosion (see **min-filter**), where the dilation and erosion are performed using the same structuring element. A morphological closing fills dark holes that are smaller than the structuring element. This operator can be applied to both binary and grey-level images.

The first parameter of this filter specifies the shape of the structuring element (neighbourhood). When set to **ball**, circular (2D) or spherical (3D) structuring elements are used. When set to **box**, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the **radius** parameter.

dilation-reconstruction [-n 4,6 | 8,26] <mask>

This operator performs the reconstruction by dilation of the specified mask from the input image. The reconstruction by dilation is the geodesic dilation iterated until stability. The implementation follows the hybrid algorithm (Vincent, 1993b).

The connectivity used for propagation is set using the **-n** option. See Section 10.1 for more information about the meaning and usage of this option.

erosion-reconstruction [-n 4,6 | 8,26] <mask>

This operator performs the reconstruction by erosion of the specified mask from the input image. The reconstruction by erosion is the geodesic erosion iterated until stability. The implementation follows the hybrid algorithm (Vincent, 1993b).

The connectivity used for propagation is set using the **-n** option. See Section 10.1 for more information about the meaning and usage of this option.

extended-maxima [-n 4,6 | 8,26] <h>

This operator computes the extended maxima of the input image. The extended maxima are the regional maxima of the h-maxima transform of the image. This transform labels bright domains that are significant in the sense that their height is equal or larger than specified **h** value.

The connectivity used in the algorithm is set using the **-n** option. See Section 10.1 for more information about the meaning and usage of this option.

extended-minima [-n 4,6 | 8,26] <h>

This operator computes the extended minima of the input image. The extended minima are the regional minima of the h-minima transform of the image. This transform labels dark domains that are significant in the sense that their depth is equal or larger than the specified **h** value.

The connectivity used in the algorithm is set using the **-n** option. See [Section 10.1](#) for more information about the meaning and usage of this option.

h-maxima [-n 4,6 | 8,26] <h>

This operator computes the h-maxima transform of the input image. This transform filters out all peaks with height smaller than the specified **h** value. Other peaks are preserved, but their altitude is decreased by **h**.

Note. The h-maxima transform of an image f is the geodesic reconstruction by dilation of f from $f - h$: $\text{h-maxima}(f) = R_f^\delta(f - h)$. The implementation of the corresponding algorithm takes care of possible numerical overflows when computing the difference $f - h$.

The connectivity used in the reconstruction algorithm is set using the **-n** option. See [Section 10.1](#) for more information about the meaning and usage of this option.

h-minima [-n 4,6 | 8,26] <h>

This operator computes the h-minima transform of the input image. This transform filters out all troughs with depth smaller than the specified **h** value. Other troughs are preserved, but their depth is decreased by **h**.

Note. The h-minima transform of an image f is the geodesic reconstruction by erosion of f from $f + h$: $\text{h-minima}(f) = R_f^\varepsilon(f + h)$. The implementation of the corresponding algorithm takes care of possible numerical overflows when computing the sum $f + h$.

The connectivity used in the reconstruction algorithm is set using the **-n** option. See [Section 10.1](#) for more information about the meaning and usage of this option.

max-filter ball | box <radius>

This operator performs a morphological dilation on the input image using a flat structuring element. This corresponds to a local maximum filter, where each position is assigned the largest value observed over its neighbourhood. This operator can be applied to both binary and grey-level images.

The first parameter of this filter specifies the shape of the structuring element (neighbourhood). When set to `ball`, circular (2D) or spherical (3D) structuring elements are used. When set to `box`, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the `radius` parameter.

min-filter ball | box <radius>

This operator performs a morphological erosion on the input image using a flat structuring element. This corresponds to a local minimum filter, where each position is assigned the smallest value observed over its neighbourhood. This operator can be applied to both binary and grey-level images.

The first parameter of this filter specifies the shape of the structuring element (neighbourhood). When set to `ball`, circular (2D) or spherical (3D) structuring elements are used. When set to `box`, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the `radius` parameter.

opening ball | box <radius>

This operator performs a morphological opening on the input image. Opening is a combination of an erosion (see `min-filter`) followed by a dilation (see `max-filter`), where the dilation and erosion are performed using the same structuring element. A morphological opening removes bright objects that are smaller than the structuring element. This operator can be applied to both binary and grey-level images.

The first parameter of this filter specifies the shape of the structuring element (neighbourhood). When set to `ball`, circular (2D) or spherical (3D) structuring elements are used. When set to `box`, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the `radius` parameter.

regional-maxima [-n 4,6 | 8,26]

This operator computes the regional maxima in the input image. A regional maximum is a connected set of positions that share the same value and for which neighbours not included in the set have strictly smaller values.

In the output image, positions not belonging to regional maxima are set to the lowest value of the image numerical type. Positions in regional maxima keep their original (input) value. This behaviour may change in the future.

The connectivity used when computing connected components is set using the `-n` option. See [Section 10.1](#) for more information about the meaning and usage of this option.

regional-minima [-n 4,6 | 8,26]

This operator computes the regional minima in the input image. A regional minimum is a connected set of positions that share the same value and for which neighbours not included in the set have strictly higher values.

In the output image, positions not belonging to regional minima are set to 0. Positions in regional minima are assigned the maximum value of the image numerical type. This behaviour may change in the future.

The connectivity used when computing connected components is set using the `-n` option. See [Section 10.1](#) for more information about the meaning and usage of this option.

size-closing [-n 4,6 | 8,26] <size>

This operator performs a closing by size attribute on the input image. When applied to a binary image, this operator fills all the background components with size below the specified threshold. When applied to a grey-scale image, this operator assigns to each position the smallest value at which the position still belongs to a connected trough component of size equal or above the specified threshold. This operator is in particular useful to filter images containing thin or elongated structures (Vincent, 1993a).

The implemented algorithm is a corrected but close to literal version of the union-find algorithm described in (Meijster and Wilkinson, 2002).

The connectivity used for propagation is set using the `-n` option. See Section 10.1 for more information about the meaning and usage of this option.

size-opening [-n 4,6 | 8,26] <size>

This operator performs an opening by size attribute on the input image. When applied to a binary image, this operator selects all the components with size equal or above the specified threshold. When applied to a grey-scale image, this operator assigns to each position the highest value at which the position still belongs to a connected peak component of size equal or above the specified threshold. This operator is in particular useful to filter images containing thin or elongated structures (Vincent, 1993a).

The implemented algorithm is a corrected but close to literal version of the union-find algorithm described in (Meijster and Wilkinson, 2002).

The connectivity used for propagation is set using the `-n` option. See Section 10.1 for more information about the meaning and usage of this option.

toggle-filter [-i <num-iterations>] ball | box <radius>

This filter performs local contrast enhancement using mathematical morphology operators (Kramer and Bruckner, 1975). Each position is assigned its corresponding value in the eroded or in the dilated image, depending on which one is the closest to its original (input) value.

The first parameter of this filter specifies the shape of the structuring element (neighbourhood) used in the dilation and erosion operations. When set to **ball**, circular (2D) or spherical (3D) structuring elements are used. When set to **box**, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the **radius** parameter.

By default, the filter is applied only once on the image. The `-i` option can be used to apply the filter recursively an arbitrary number of times. Increasing the number of iterations reinforces the effect of the filter.

tophat-black ball | box <radius>

This operator applies the top-hat transform (“black” version) to the input image, obtained by subtracting the original image to the result of its closing by a structuring element. This operation achieves a background removal in situations where the background is brighter (higher intensities) than the objects of interest. The structuring element should be taken as least as large as the objects of interest. Objects larger than the structuring element will be lumped into the background by this operation.

The first parameter of this operator specifies the shape of the structuring element (neighbourhood) used in the opening operation. When set to `ball`, circular (2D) or spherical (3D) structuring elements are used. When set to `box`, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the `radius` parameter.

tophat-white ball | box <radius>

This operator applies the top-hat transform (“white” version) to the input image, obtained by subtracting to the original image the result of its opening by a structuring element. This operation achieves a background removal in situations where the background is darker (lower intensities) than the objects of interest. The structuring element should be taken as least as large as the objects of interest. Objects larger than the structuring element will be lumped into the background by this operation.

The first parameter of this operator specifies the shape of the structuring element (neighbourhood) used in the opening operation. When set to `ball`, circular (2D) or spherical (3D) structuring elements are used. When set to `box`, square (2D) or cubic (3D) structuring elements are used. The size of the structuring element is controlled by the `radius` parameter.

8 Analysis operators

8.1 Label operators

`clean-borders [-l]`

This operator removes the image contents touching the borders. This is obtained by subtracting from the input image its morphological reconstruction from the border. The operator can be applied to both intensity and label images; however, its typical application is the removal of segmented objects that intersect the image border.

The `-l` option must be used when processing label images.

`click-select [-d max-distance] <positions.vx>`

This operator selects labels in the input image based on specified positions. The algorithm works by determining for each position the closest label, i.e. the label having the pixel/voxel closest to the specified position. Proximity is measured using Euclidean distance, taking into account the spatial calibration of the image.

The file `<positions.vx>` contains the positions to be used for the selection, following the `sviewer` in-house shape file format. The positions should be defined in the physical space of the image.

The `-d` option allows to set a maximum distance, above which labels are simply ignored.

`click-replace [-d max-distance] <positions.vx> <new-value>`

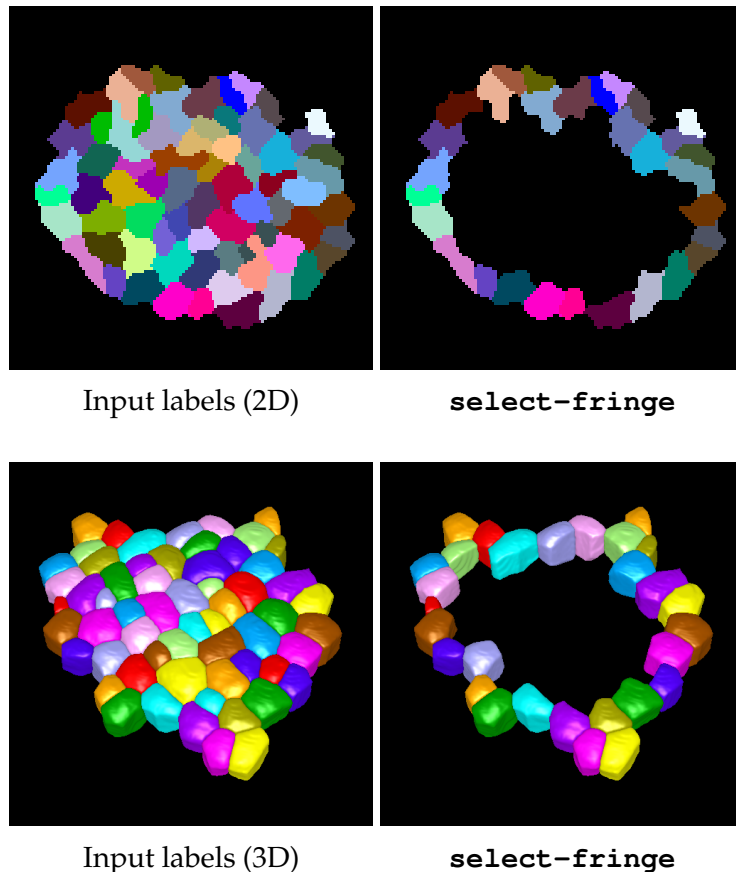
This operator replace labels in the input image based on specified positions. The algorithm works by determining for each position the closest label, i.e. the label having the pixel/voxel closest to the specified position, and replaces this label by a new value. Proximity is measured using Euclidean distance, taking into account the spatial calibration of the image.

The file `<positions.vx>` contains the positions to be used for the selection, following the `sviewer` in-house shape file format. The positions should be defined in the physical space of the image. The parameter `<new-value>` specifies the value to assign to the selected labels.

The `-d` option allows to set a maximum distance, above which labels are simply ignored.

`label-boundaries [--add]`

This operator computes the boundaries between labels in an image. By default, the



split [-h <tolerance>]

This operator separates objects that are touching each other in a binary image. The optional tolerance parameter can be set to control the degree of separation (default tolerance value is 0.5). The higher the tolerance, the less the objects are split (Figure 12).

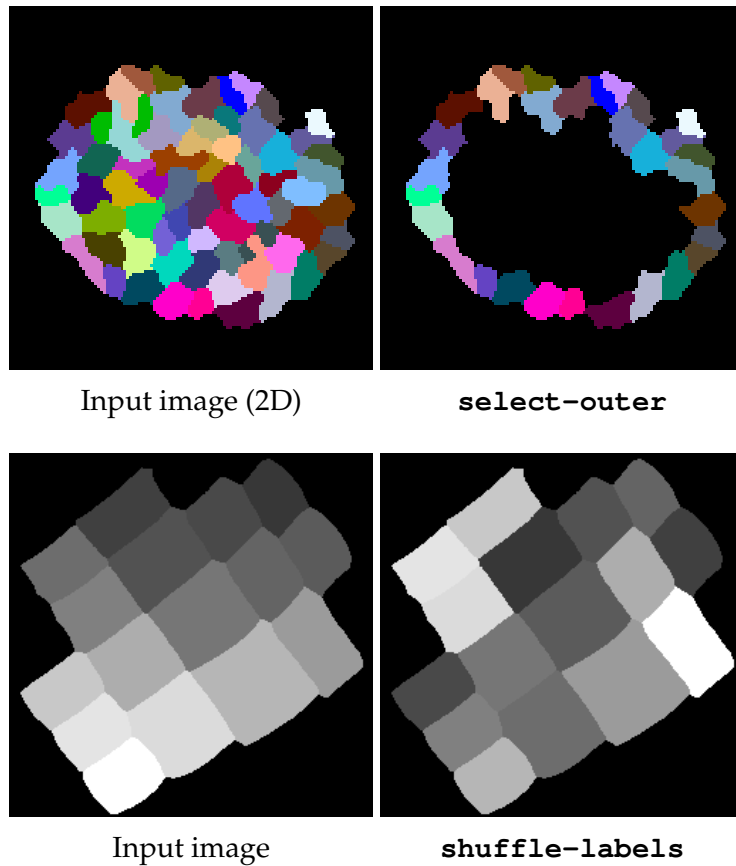
The **split** operator can also process label images of segmented objects. In this case, the original labelling is lost because a thresholding of values above 0 is applied as a first step in this operator.

swap-labels <list1> <list2>

This operator swaps values in an image. The values to swap are specified by two lists of comma-separated values. The two lists must have the same size. All image positions having a value from **list1** are assigned the corresponding value in **list2**, and reciprocally.

label-defragmentation [-n 4,6 | 8,26]

This operator takes as input a label image in which each label may appear as several connected components. The operator retains the largest component for each label and removes all the other ones. The discarded components are removed by filling the holes they make in the largest components of other labels.



The connectivity used to define connected sets is set using the `-n` option. See [Section 10.1](#) for more information about the meaning and usage of this option.

`zmap [-1]`

This operator performs a Z-front detection in the input 3D label image. Z-front detection is a 2D projection of the 3D image: at each XY position, it determines and retains the first Z position with a non-null value. Used in conjunction with the reverse operator `izmap`, this operator is typically used to extract the uppermost layer in a label image.

If the `-1` option is set (“label mode”), then the operator retains the first encountered label rather than its Z position.

8.2 Distance maps

`chamfer-distance <foreground>`

This operator computes an integer-valued map of the approximate distance between each image position and the closest position with value *foreground*. The input image is not necessarily binary, as every non-foreground position will be considered as background. The implementation follows algorithms given by Borgefors ([Borgefors, 1986](#);

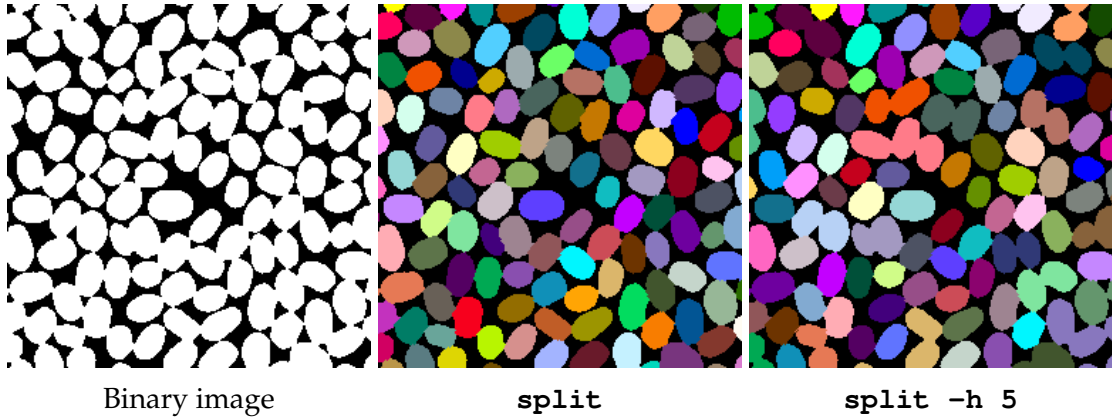


Figure 12: Separating touching objects using the `split` operator.

Borgefors, 1996).

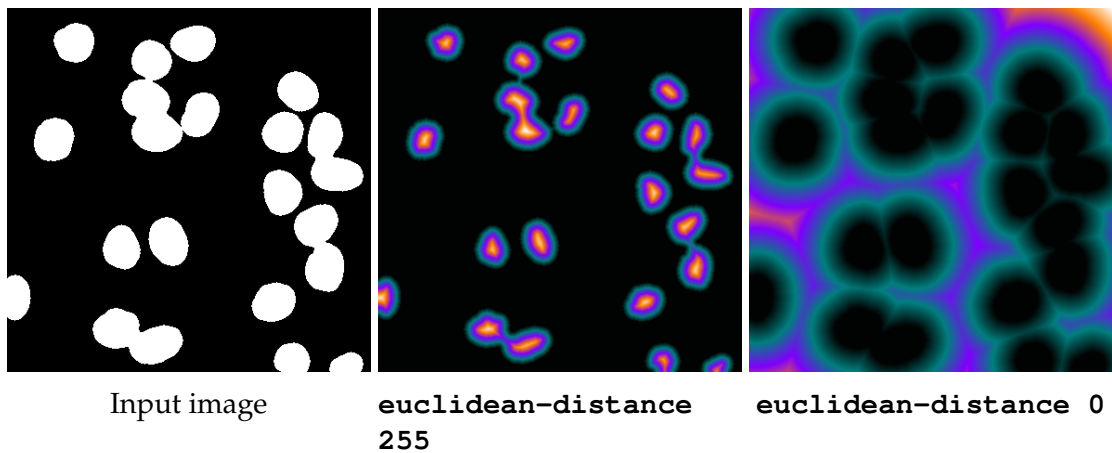
This operator is provided for historical reasons mainly. Users will in general opt instead for the Euclidean distance operator, which is very efficient and provides exact values.

`euclidean-distance [--physical] <foreground>`

This operator computes a real-valued map of the exact Euclidean distance between each image position and the closest position with value *foreground*. The input image is not necessarily binary, as every non-foreground position will be considered as background. The implementation is based on Felzenszwalb and Huttenlocher's algorithm (Felzenszwalb and Huttenlocher, 2012) that we have modified to take into account the spatial calibration of the image.

By default, the distance is computed in pixels (2D) or voxels (3D). If the `--physical` option is set, the distance is instead computed in physical units, taking into account the spatial calibration of the image and its anisotropy, if any.

The numerical type of the output image is 32-bit floating point.



8.3 Measurements

`region-analysis [-p parameter[,parameter2,...,parameterN]] implicit | explicit`

This operator performs quantitative measurements on objects in a labelled image. Each label in the input image is considered as defining an object. In practice, an object will generally correspond to a unique connected component in the input label image, but BIP imposes no restriction on the number of components per object.

The only mandatory parameter of the operator is the boundary mode. The **implicit** mode must be used if there is no specific labeling of object boundaries in the segmented images to process. This corresponds to the situation when different object labels may be in contact. If a specific label has been used to label boundaries at the segmentation stage (as done for example when computing the dams in a watershed transform), then the **explicit** mode should be used. Support for this mode is incomplete at the moment, as there is no control of the boundary value. The boundary mode specification will likely change in the future.

By default, BIP performs measurements for all implemented parameters. This behaviour can be overridden by using the **-p** option to specify a comma-separated list of desired parameters. For example, the following call will only measure the number of voxels, the volume and the sphericity of objects in a 3D image:

```
shell$ bip region-analysis -p count,volume,sphericity implicit \
image.tif
```

Available parameters in BIP fall into one out of three categories. Most parameters are simple parameters, for which a single numerical value is obtained (e.g., **area**). [Table 3](#) lists the available simple parameters. Some other parameters are vector parameters, which provide several values corresponding to as many simple parameters that are components of a vector (such as a position or a direction). One example is the **centroid** parameter, which provides on output 2 or 3 parameters (**centroid-x**, **centroid-y** and **centroid-z**), corresponding to the coordinates of the average position of an object. Lastly, group parameters are short-cuts for specifying groups of related simple or vector parameters, without having to specify each parameter individually. For example, asking for the **size** parameter triggers the measurements of the **count**, **area**, and **volume** parameters. [Table 4](#) lists the available vector and group parameters.

On output, BIP writes a **region-analysis.df** file, which obeys a tsv format (columns of measured values separated by tabulations). This file can be loaded in software such as Excel, LibreOffice Calc, gnuplot, or R for downstream statistical analysis and graphical plotting.

As for most BIP operators, both 2D and 3D images can be passed upon a same call to the **region-analysis** operator. Some parameters are defined in both 2 and 3 dimensions (such as **count**, the number of times the label is observed in the image), while

others are specific to either the 2D (such as **area**) or the 3D case (such as **volume**). BIP reports a **NaN** (not-a-number) value in the output file for undefined parameters.

Images with multiple channels and/or multiple timepoints can be passed to the operator, in which case labelled regions will be quantified in the different channels and time frames. A **channel** column is added in the output file in case the input image contains several channels. Similarly, a **timepoint** column is added in the output file in case the input image contains several timepoints. Images with different numbers of channels or timepoints can be mixed during a same call.

8.4 Export operators

export-bboxes

This operator computes the bounding boxes of the labelled regions in the input image. The coordinates of the bounding boxes are expressed in the physical space (i.e., taking into account the spatial calibration of the image). For each label, the corresponding box is stored in a file named by appending the label to the input filename. The channel and the timepoint are also added to the filename if the input image contains several channels or timepoints. The output file format is Free-D's shape viewer format.

export-shapes [--stl]

This operator computes the triangular meshes corresponding to the boundaries of labelled regions in a 3D image. One mesh is computed for each label and stored in a file named by appending the label to the input image filename. The mesh is obtained by applying the Marching Cubes algorithm (Lorensen and Cline, 1987) to the binary mask of the label.

The default output file format is the svviewer shape file format (*.tm* files). You can switch to the STL file format using the **--stl** option.

export-rag

This operator computes the region adjacency graph of the input image and saves it in a format suitable for opening with svviewer, the Free-D's standalone 3D shape viewer.

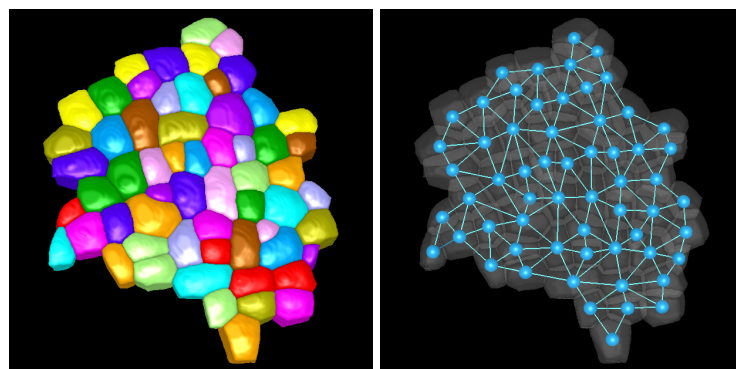
In the RAG, each labelled region of the input image is represented by a node placed at the centroid (geometrical center) of the region. The nodes that correspond to neighbouring regions are connected by edges.

Parameter	Definition	Dimension
<i>Size measurements</i>		
area	Object area in physical units	2D
count	Number of pixels (2D) or voxels (3D)	2D,3D
volume	Object volume in physical units	3D
equiv-radius	Radius of disk (or sphere) of same area (volume)	2D,3D
major-radius	Radius along the first principal axis	2D,3D
medium-radius	Radius along the 2nd principal axis	3D
minor-radius	Radius along the smallest principal axis	2D,3D
gyration-radius	Root mean squared distance to object centroid	2D,3D
perimeter	Length of the object boundary	2D
surface-area	Surface area of the object boundary	3D
box-diagonal	Diagonal of the axis-aligned bounding box	2D,3D
<i>Position measurements</i>		
centroid-x	Centroid: x-coordinate	2D,3D
centroid-y	Centroid: y-coordinate	2D,3D
centroid-z	Centroid: z-coordinate	3D
box-x1	Axis-aligned bounding box: x-coordinate of 1st corner	2D,3D
box-y1	Axis-aligned bounding box: y-coordinate of 1st corner	2D,3D
box-z1	Axis-aligned bounding box: z-coordinate of 1st corner	3D
box-x2	Axis-aligned bounding box: x-coordinate of 2nd corner	2D,3D
box-y2	Axis-aligned bounding box: y-coordinate of 2nd corner	2D,3D
box-z2	Axis-aligned bounding box: z-coordinate of 2nd corner	3D
<i>Orientation measurements</i>		
major-vector-x	Largest principal axis: x-coordinate	2D,3D
major-vector-y	Largest principal axis: y-coordinate	2D,3D
major-vector-z	Largest principal axis: z-coordinate	3D
minor-vector-x	Smallest principal axis: x-coordinate	2D,3D
minor-vector-y	Smallest principal axis: y-coordinate	2D,3D
minor-vector-z	Smallest principal axis: z-coordinate	3D
medium-vector-x	Intermediate principal axis: x-coordinate	3D
medium-vector-y	Intermediate principal axis: y-coordinate	3D
medium-vector-z	Intermediate principal axis: z-coordinate	3D
<i>Shape measurements</i>		
elongation	Length-ratio between largest and intermediate axes	2D,3D
flatness	Length-ratio between intermediate and smallest axes	3D
circularity	Matching with a disk having the same area	2D
sphericity	Matching with a sphere having the same volume	3D
globularity	(Experimental)	2D,3D
<i>Topological measurements</i>		
degree	Number of neighbouring labels	2D,3D
outer	Indicator of contact with the background	2D,3D
fringe	Indicator of localization at the periphery of the tissue	2D,3D
layer	Number of the containing tissue layer	2D,3D

Table 3: Simple parameters available in **BP**. Dimension indicates in which dimension(s) each parameter is defined. Parameters not defined for a given dimension will appear as **NaN** in the measurement file.

Composiite parameter	Definition	Associated simple parameters
centroid	Object average position	centroid-x, centroid-y, centroid-z
size	Object size	count, area, volume
box	Axis-aligned bounding box	box-x1, box-y1, box-z1, box-x2, box-y2, box-z2, box-diagonal

Table 4: Composite and meta-parameters available in BIP.



Input labels (3D)

export-rag

9 Pipelines

BIP pipelines have two purposes. First, they simplify and accelerate the design and the application of image processing sequences composed of several steps. Second, they also facilitate tracing the operations that have been applied to images.

9.1 Writing and using pipeline files

Consider the following 3-step sequence: Gaussian filtering, Otsu's thresholding, and object labelling. Without pipelines, the corresponding commands to enter into the terminal would be:

```
shell$ bip gaussian-filter 1.0 image.tif
shell$ bip otsu-thresholding image-gaussian-filter.tif
shell$ bip labelling image-gaussian-filter-otsu-thresholding.tif
```

This is long, both to type and to execute because of writing/reading operations of intermediate files. This is also error-prone when applying this sequence to different files and at different times. In addition, though the intermediate files generated by such a sequence may be of interest at the design stage, they are generally of no interest in routine application. Using a pipeline can address all these issues at once.

To use a pipeline with BIP, you first create a text file containing the sequence of operations to apply. The file can also contain comments (pieces of text starting with the '#' character, which are ignored by BIP). For the above sequence, one would write the following code and save it to a file named, for example, **process.pipeline**:

```
# contents of process.pipeline
gaussian-filter 1.0
otsu-thresholding
labelling
```

Importantly, note how the syntax is identical to the one used when invoking operators one-by-one on the command line. This feature enables users to rapidly write pipelines as soon as they have learn to use BIP in simple command-line mode.

To apply a pipeline, one invokes the **pipeline** operator with the corresponding file:

```
shell$ bip pipeline process.pipeline image.tif
```

When running a pipeline, the name of the output file for each input image is obtained by concatenating the input image filename with the basename of the pipeline file. In our example, the result would be stored in **image-process.tif**.

Of course, pipelines can be applied in batch as any other operator:

```
shell$ bip pipeline process.pipeline ../input/*.tif
```

9.2 Pipelines without pipeline files

In some applications, it may happen that short pipelines have to be applied and/or that keeping trace of the applied pipelines into text files is not required. In these cases, writing pipeline files may be cumbersome. BIP allows defining pipelines upon invocation of the **pipeline** operator.

Pipeline expressions are used to define pipelines upon invocation of the **pipeline** operator in the command-line. As in pipeline files, the syntax in pipeline expressions is identical to the syntax used to invoke operators individually. The only specificity in expression syntax is that the successive operators are separated by the '|' symbol (a.k.a. pipe operator under Unix/Linux). The whole expression should be enclosed within a pair of quotes.

The **-e** option of the **pipeline** operator is used to specify a pipeline expression instead of a pipeline file. Using a pipeline expression, our sample pipeline above would thus be applied using the following call:

```
shell$ bip pipeline -e "gaussian-filter 1.0 | \
    otsu-thresholding | labelling" image.tif
```

9.3 Pipeline variables

It is frequent that a pipeline does not consist in a purely sequential set of operations but instead requires that different images undergo different operations before being combined. As a prototypical example, consider the morphological tophat, defined for an image I as:

$$\text{tophat}(I) = I - \text{opening}(I)$$

(note this is a purely pedagogical example, as BIP provides a native **tophat-white** operator that implements this operation). In a pipeline, the opening step (here, using a circular structuring element of radius 2) can be computed by writing

```
opening-filter ball 2
```

The problem is that after the execution of this pipeline line, the initial image is lost. This shows that two images are needed to compute the tophat: one to store the original image and another to store the opening of this image. Hence, we need to

create a copy of the original image that can be retrieved after having computed the opening.

This is where variables come into play. Remember that the image to which the operators are applied in a pipeline is implicit. A variable is a name that designates this current image at a given step, the step corresponding to the point where the variable is declared. A variable declaration consists of the keyword **store** followed by the variable name, preceded by the special character '\$':

```
store $<variable-name>
```

The current image in a pipeline can be set to a previously stored image by invoking the **recall** operator followed by the corresponding variable name:

```
recall $<variable-name>
```

In the tophat example, we need to create a variable to store the input (= current image at the beginning of the pipeline) image before computing its opening. We then recall the original image and subtract the opening. This yields the following pipeline:

```
# save a copy of the input image somewhere in memory  
store $image  
  
# replace current image by its morphological opening  
# and store the result somewhere in memory  
opening ball 2  
store $opening  
  
# set the current image to the previously stored input image  
recall $image  
  
# subtract the morphological opening from the current image  
subtract $opening
```

10 Advanced usage

10.1 Neighbourhood systems and connectivity

Many BIP operators have a `-n` option to set the connectivity of the neighbourhood system used. When passing `4`, `6`, or `4, 6` as a value for this option, a Von Neumann neighbourhood is used: the neighbours of a pixel or voxel are connected to it by edges parallel to the image XYZ axes (stated otherwise, each neighbour differs by only one coordinate from the center of the neighbourhood). When passing `8`, `26`, or `8, 26`, a Moore neighbourhood is used instead (diagonal pixels or voxels are also neighbours).

There is absolutely no difference between the three possibilities for specifying a neighbourhood system. For example, all three values `8`, `26`, and `8, 26` will impose 8-connectivity in 2D and 26-connectivity in 3D. The three possibilities are essentially offered for clarity and consistency. To process 2D images, one would certainly write:

```
shell$ bip labelling -n 8 image2D-A.tif image2D-B.tif
```

and to process a mix of 2D and 3D images, one may write:

```
shell$ bip labelling -n 8,26 image2D-A.tif image3D-C.tif
```

but the result would be the same by calling, for example:

```
shell$ bip labelling -n 26 image2D-A.tif image3D-C.tif
```

Lastly, note the default connectivity is 4 (2D)/6 (3D). Hence, the `4, 6`, and `4, 6` option values are essentially provided for clarity and for making explicit the choice of the neighbourhood system.

10.2 Pattern substitutions on file and directory names

Several BIP operators are binary operators: they take as input two images, the current image to process and an additional image required for the processing of the current image. A typical example is the **mask** operator, in which a mask image is used to set to zero all values in the input image that are outside the defined mask. Some operators also take as input additional files that are not images. A typical example is the **click-select** operator, which takes as input a file of selected positions.

Under a well-designed file nomenclature, any additional file (image or else) should have a name that can be automatically constructed from the filename of the currently processed image. For example, a mask could be stored in a **masks** folder under the same filename as the current image, or with a suffix such as ***-mask.tif**.

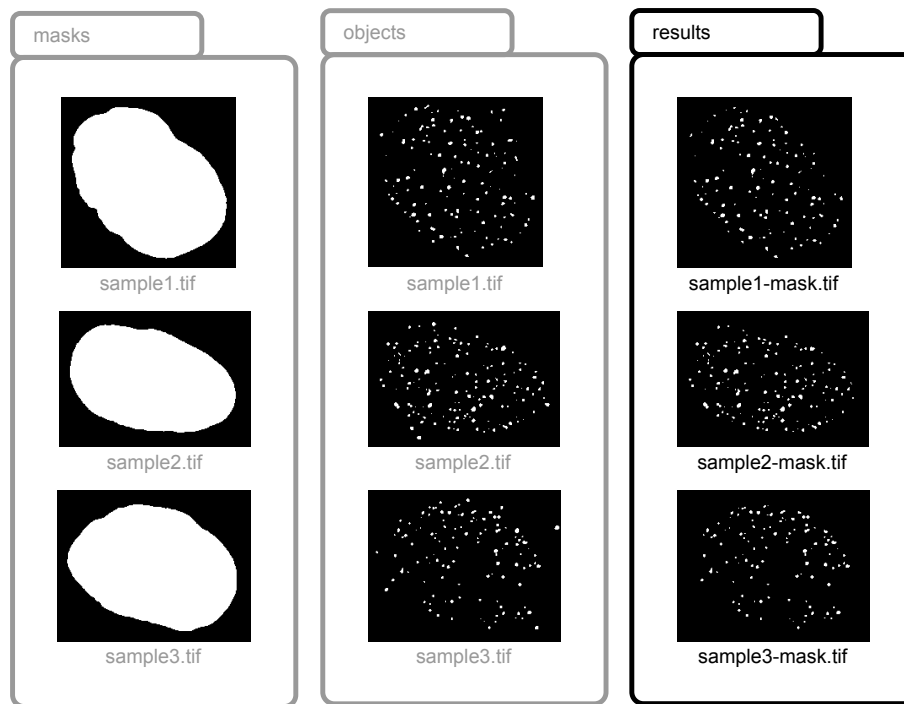


Figure 15: Matching processed image files with additional files (simple situation). Each input image in the **objects** folder has a corresponding mask image in the **masks** folder. Specifying the **masks** folder is sufficient for BIP to associate each object image to its corresponding mask. The following command was run from the **results** folder: **bip mask ../masks ../objects/***.

BIP features a pattern substitution mechanism based on regular expressions to automatically construct, from the currently processed image filename, the name of an associated additional file. Pattern substitution allows to replace parts of a filename by some specific pattern. It also allows to remove or add some parts. The position (begin, end, or anywhere) of replacement, removal, and insertion can be specified.

The simplest situation is when no substitution is needed, because additional files are stored in some directory under the same filenames as the processed images (Figure 15). It is then sufficient to specify the path to this directory. For example:

```
shell$ bip mask ../masks ../inputs/*
```

Another typical situation is when the processed images and their associated additional files correspond to different fluorescence channels indicated in filenames (e.g., with "c02" and "c01" tags; see Figure 16). In this case, the "c02" pattern in the input filenames has to be replaced by "c01" to associate the processed images to their respective additional files:

```
shell$ bip mask ../masks:s/c02/c01/ ../inputs/*
```

Note how the substitution pattern is separated by a ':' character from the directory name. The pattern itself is composed of three parts separated by slashes ('/'): (1) a

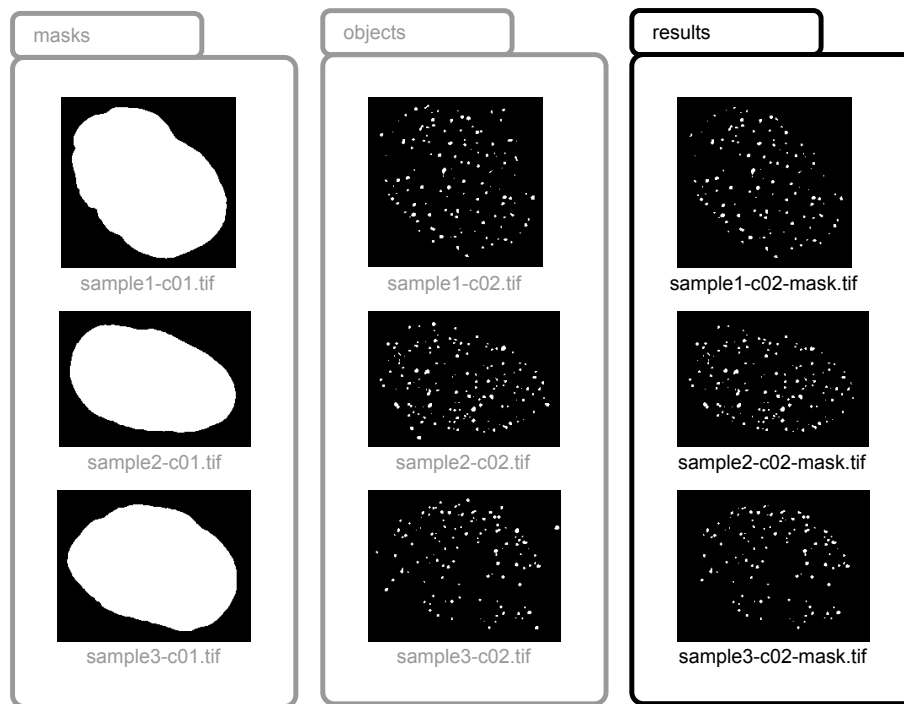


Figure 16: Matching processed image files with additional files (typical situation). Each input image in the **objects** folder has a corresponding mask image in the **masks** folder. The only difference in filenames is the channel suffix. A pattern substitution has to be given with the **masks** folder name to ensure BIP associates each object image to its corresponding mask. The following command was run from the **results** folder:

```
bip mask ../masks:s/02/01 ../objects/*.
```

letter designating an action to perform, here 's' for substitution; (2) a regular expression specifying the pattern to match, here the "c02" substring; (3) the motif with which the occurrences of the searched pattern should be replaced, here "c01".

A specific position of the matching pattern in the input filename can be specified using the '^' (begin) or '\$' (end) characters. For example, "**s/tif\$/vx/**" is used to replace the **.tif** extension of an image by the **.vx** extension of a file listing a set of positions from which to operate some operation (see, for example, **click-select**). Similarly, "**s/^/mask-/**" is used to add the prefix **mask-** to the image filename.

Several substitutions patterns can be specified at the same time, in which case each pattern should be separated from the previous one by a comma. For example, the call:

```
shell$  bip mask ../masks:s/c02/c01/,s/^/mask-/ ../inputs/*
```

would look, for each input image, for an associated image in the **masks** directory with the same filename, prefixed by **mask-** and in which the substring **c02** would be replaced by **c01**.

11 Global options

-u

This options disables compression of output files. This is useful when fast file writing (and subsequent reading) is required, given that compressing and decompressing data upon file writing and reading induces a computational cost that can be significant with large images. This option is also useful when writing images that are intended to be loaded into other software that, like ImageJ/Fiji, are not very efficient in loading compressed TIF files.

-f

This option allows BIP to overwrite existing files. This is useful when calling BIP repeatedly on the same inputs, for example when testing different pipelines or operator values. By default, BIP will not overwrite any existing file and will generate an error when attempting to write over an existing file.

In the example below, the second call generates an error because an output file has already been created in the current directory by the first call:

```
shell$ bip gaussian-filter 1.0 ../input/image.tif
shell$ bip gaussian-filter 1.0 ../input/image.tif
shell$ *** error: Exception raised:
shell$   What: File already exists
shell$   Where: void ImageWriter::write(...)
shell$   Calls: [0] void ImageWriter::write(...)
shell$   Filename: image-gaussian-filter.tif
```

With the `-f` option, the file created upon the first call is silently overwritten:

```
shell$ bip gaussian-filter 1.0 ../input/image.tif
shell$ bip -f gaussian-filter 1.0 ../input/image.tif
```

-b, --no-suffix

This option removes the suffix (name of applied operator) that is automatically added to filenames when writing output files (`-b` stands for “bare”). Hence, using this option results in having identical filenames for input and for output images. Beware that using this option in conjunction with the `-f` option can be very dangerous, as permanent loss of files may result. To avoid loosing precious data, it is highly recommended to never run BIP within a folder containing input images.

--long-suffix

This option asks BIP to use long suffixes instead of just operator names as suffixes. Long suffixes typically contain values of operator values. This is useful for example to

store the results obtained with different parameter settings for a given operator. Without this option, only one output could be stored without further file manipulation.

For example, the second call below generates an error:

```
shell$ bip median-filter ball 1 ../input/image.tif
shell$ bip median-filter ball 2 ../input/image.tif
shell$ *** error: Exception raised:
shell$   What: File already exists
shell$   Where: void ImageWriter::write(...)
shell$   Calls: [0] void ImageWriter::write(...)
shell$   Filename: image-median-filter.tif
```

Using `--long-suffix`, no error is generated:

```
shell$ bip --long-suffix median-filter ball 1 ../input/image.tif
shell$ bip --long-suffix median-filter ball 2 ../input/image.tif
```

Indeed, two distinct files have been created:

```
image-median-filter-ball-1.tif
image-median-filter-ball-2.tif
```

`--gen-completion-script`

(Linux) With this option set, BIP generates a bash completion script and immediately returns. The generated file is to be moved to, or copied into, the `.bash_completion` file in the user home directory. This provides the tab-completion functionality on BIP operator names.

`-B <background>`

This option sets the background value for many operators that rely on a distinction between background and foreground. Examples include thresholding operators and operators for binary mathematical morphology.

The default background value is 0.

`-F <foreground>`

This option sets the foreground value for many operators that rely on a distinction between background and foreground. Examples include thresholding operators and operators for binary mathematical morphology.

The default foreground value is 1.

12 Project-specific operators

live-track <config> <target>

This operator computes the registration of the input image on the specified target image. This operator is designed to be invoked from macros or scripts in Zen software on Zeiss microscopes during time-lapse acquisitions to allow adjusting the position of the microscope stage during acquisition.

The registration is performed using a simple translation. The optimal translation is determined by maximizing the cross-correlation between the input image and the target image. No interpolation is performed on the cross-correlation, so that the raw translation components are integer values.

The `config` file specifies the configuration to be used for the registration. This file follows the `json` format. The minimum configuration contains the following tags:

```
{
  "pipeline" : "default",
  "source-channel" : 0,
  "target-channel" : 0,
  "source-timepoint" : 0,
  "target-timepoint" : 0
}
```

The `pipeline` tag is used to specify the preprocessing pipeline to apply to the input images before computing the registration. This typically includes projection from 3D stacks to 2D images and noise filtering. The name of the pipeline does not refer to a text file (as in the `pipeline` operator) but designates instead a hard-coded sequence of operations. The only possible value at the moment for the `pipeline` tag is `default`, which is adapted to the preprocessing of fluorescently labelled growing roots. Other values will be added in the future to select other processing pipelines adapted to other organs and imaging conditions. It is likely that in future versions, registration criteria and registration algorithms will differ between pipelines, as cross-correlation maximization may not cover the full range of targeted applications.

The `source-channel` and `target-channel` tags specify the indexes of the image channels to use for computing the registration. They should be set to 0 for mono-channel images. Note the channels may differ between source and target. Though it is not expected to be useful in many situations, this feature is introduced to support flexibility in future applications.

The `source-timepoint` and `target-timepoint` tags specify the indexes of the image timepoints to use for computing the registration. If images acquired at different timepoints during a time-lapse acquisition are stored in separate files containing each a single timepoint, these attributes will be set to 0. If the images are stored in a single file containing all timepoints, then these attributes will typically take consecutive

values (first call with 1 and 0, second with 2 and 1, etc.). However, non-consecutive values can be specified in case registration is invoked at arbitrary intervals across acquisitions instead of being called upon each acquisition.

The operator outputs a `.json` file reporting the components of the computed translation (in both logical and physical units), the output status as well as encountered errors, if any. For example, with channels and timepoints set to 0 in the configuration file, invoking BIP with the command:

```
shell$ bip live-track config.json target.tif source.tif
```

will generate on output a file `source-live-track.json` with the following contents:

```
{
  "config" : {
    "file" : "config.json",
    "pipeline" : "default",
    "source-channel" : 0,
    "source-timepoint" : 0,
    "target-channel" : 0,
    "target-timepoint" : 0
  },
  "date" : "2021.10.21_(18:56:45)",
  "images" : {
    "source" : "target.tif",
    "target" : "source.tif"
  },
  "registration" : {
    "elapsed-time-msec" : 38,
    "physical-horizontal-shift" : 2.905931,
    "physical-units" : "micrometer",
    "physical-vertical-shift" : 0.415133,
    "raw-horizontal-shift" : 7,
    "raw-vertical-shift" : 1
  },
  "report" : {
    "error" : "none",
    "error-message" : "",
    "status" : "ok"
  }
}
```

13 BIP in action: illustrated examples

The examples below illustrate how diverse tasks of varying complexity can be performed by combining BIP operators (typically using pipelines). Some examples are of purely pedagogical interest (for example, when a BIP operator already implements the task at hand). Others correspond to typical, recurrent needs that are frequently encountered in bioimage analysis studies. To maximise the benefits of reading this section, we encourage readers to consider these examples as “exercises” and to try building their own solutions before reading the proposed solutions.

13.1 Black tophat

Problem. Write down the sequence of operations corresponding to the black tophat computed with a circular structuring element of radius 3. For an image I , the black tophat is defined by:

$$\text{tophat}(I) = \text{closing}(I) - I$$

Also write the corresponding pipeline. [Note this is a purely pedagogical problem since BIP already comes with a **tophat-black** operator.]

Answer. There are two instructions, the first one to compute the closing of the input image and the second one to subtract the image from the result of the closing operation:

```
shell$ bip closing-filter ball 3 image.tif
shell$ bip subtract image.tif image-closing-filter.tif
```

Note the order of the arguments in the second instruction: `image.tif` is the argument of the `subtract` operator, which is applied to `image-closing-filter.tif`.

The corresponding pipeline requires the creation of a variable to store the input image before it is modified by the closing operation. This allows to recover the input image to subtract it from the result of the closing operation:

```
store $image
closing-filter ball 3
subtract $image
```

13.2 Selecting objects based on size

Problem. Find the sequence of operations to remove in a segmented image all the objects that are below some size threshold (taking the number of pixels as a size measurement). Then write the corresponding pipeline file.

Answer. The principle is to first generate a map of object size and then threshold the map at the desired minimum size. This yields the following two instructions:

```
shell$ bip map-parameter area labels.tif
shell$ bip threshold 500 labels-area.tif
```

This gives a binary image of the objects with a size above the threshold of 500 pixels. Labelling could then be applied to relabel the selected objects:

```
shell$ bip labelling labels-area-threshold.tif
```

There are however two potential problems when proceeding this way. First, the obtained labels will generally differ from the labels originally stored in **labels.tif**. This can be an issue if measurements are to be made on both the original and the filtered images and combined later on a per label basis. Second, objects that are touching in the original image would be merged into a single label.

Hence, the proper way to obtain a label image of objects larger than the size threshold is to mask the original label image with the size-thresholded image:

```
shell$ bip mask labels-area-threshold.tif labels.tif
```

Putting it all together into a single pipeline file gives:

```
# keep a copy of labels for final masking
store $label-image

# threshold objects based on area
map-parameter area
threshold 500
store $binary-mask

# mask original labels with mask of selected objects
recall $label-image
mask $binary-mask
```

13.3 Selecting objects based on size and shape

Problem. This is a generalization of the problem of [Section 13.2](#): find the sequence of operations to remove in a segmented image all the objects that are below some size threshold (taking the number of pixels as a size measurement) and below an elongation threshold. Then write the corresponding pipeline file.

Answer. The principle is to first generate a map of object size and then threshold the map at the desired minimum size. Same is performed for elongation. The resulting

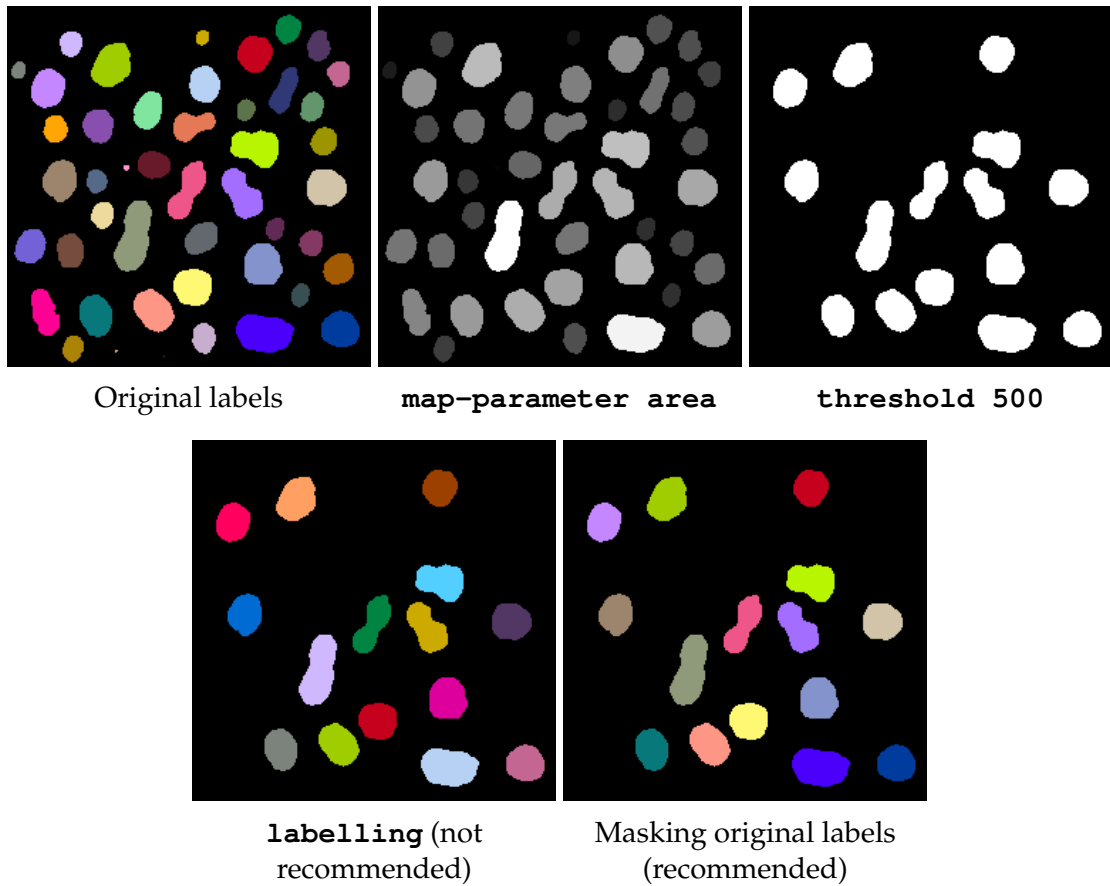


Figure 17: Selecting objects based on size (see [Section 13.2](#)).

two masks are used to filter in cascade the original label image (note they could also have been combined into a unique mask applied once to the labels):

```
shell$ bip map-parameter area labels.tif
shell$ bip map-parameter elongation labels.tif
shell$ bip threshold 500 labels-area.tif
shell$ bip threshold 1.5 labels-elongation.tif
shell$ bip mask labels-area-threshold.tif labels.tif
shell$ bip mask labels-elongation-threshold.tif labels-mask.tif
```

The corresponding pipeline is:

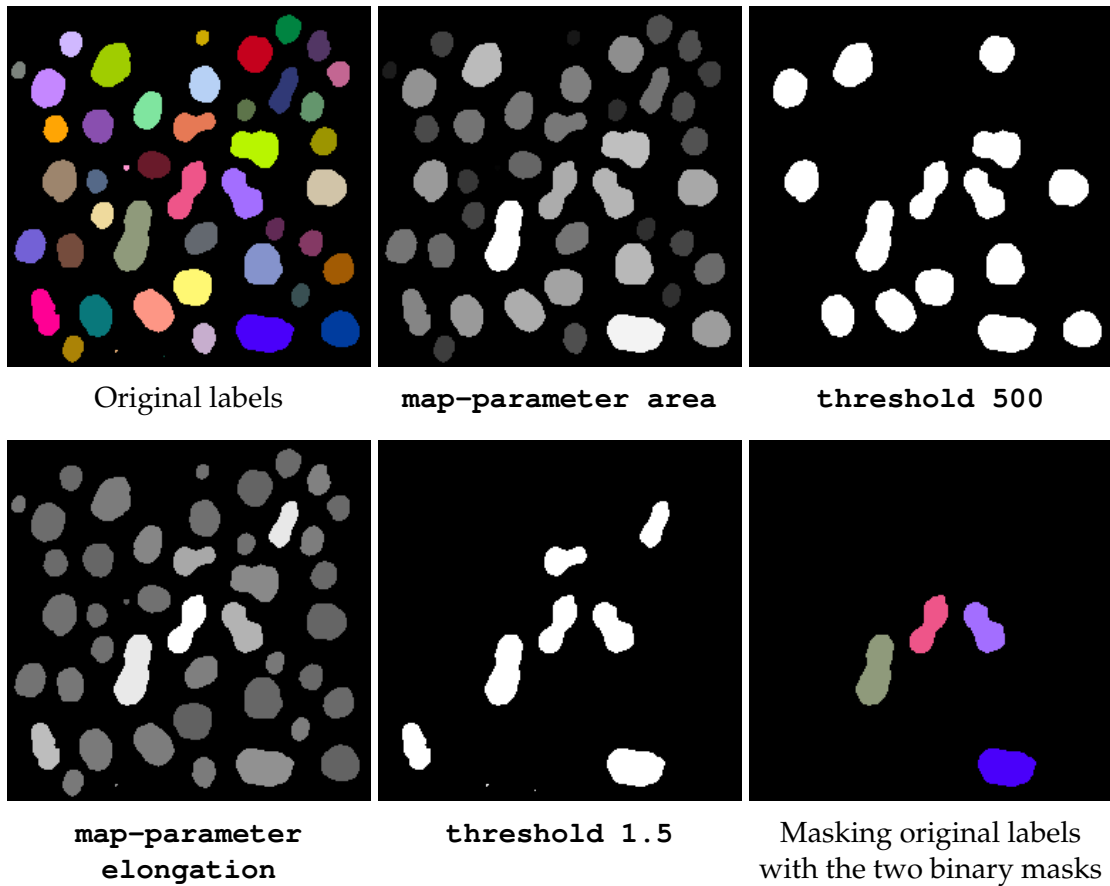


Figure 18: Selecting objects based on size and shape (see [Section 13.3](#)).

```

# keep a copy of labels for final masking
store $label-image

# threshold objects based on area
map-parameter area
threshold 500
store $area-mask

# threshold objects based on elongation
recall $label-image
map-parameter elongation
threshold 1.5
store $elongation-mask

# mask original labels with the two binary masks
recall $label-image
mask $area-mask
mask $elongation-mask

```

13.4 Labelling clusters of neighbouring objects

Problem. Starting from an image of labelled objects, find the sequence of operators that assign a same label to objects that are close from each other (consider for example a distance below 5 units of physical distance). Then write the corresponding pipeline.

Answer. The principle is to dilate objects so that neighbours touch each other. The resulting aggregates are labelled and the original object shapes are recovered using masking by the input label image.

Note the dilation is performed below by thresholding the background distance map rather than by simply calling the dilation operator. This allows to handle images with non-cubic voxels (or non-square pixels), thanks to the **--physical** option of the Euclidean distance map operator (as done in the pipeline version below).

```
shell$ bip threshold 1 labels.tif
shell$ mv labels-threshold.tif tmp.tif
shell$ bip -f --no-suffix euclidean-distance 0 tmp.tif
shell$ bip -f --no-suffix threshold 5 tmp.tif
shell$ bip -f --no-suffix invert auto tmp.tif
shell$ bip -f --no-suffix labelling tmp.tif
shell$ bip -f --no-suffix mask labels.tif tmp.tif
```

Tip: note how we repeatedly used a single intermediate image *tmp.tif* to store the successive steps in this sequence, thanks to the **-f** and **--no-suffix** options. This prevents generating multiple files with long filenames.

The corresponding pipeline is the following:

```
# keep a copy of input labels
store $input

# dilate objects with R=5 physical units
threshold 1
euclidean-distance --physical 0
threshold 5
invert auto

# label clusters of objects
labelling

# recover original, individual object shapes
mask $input
```

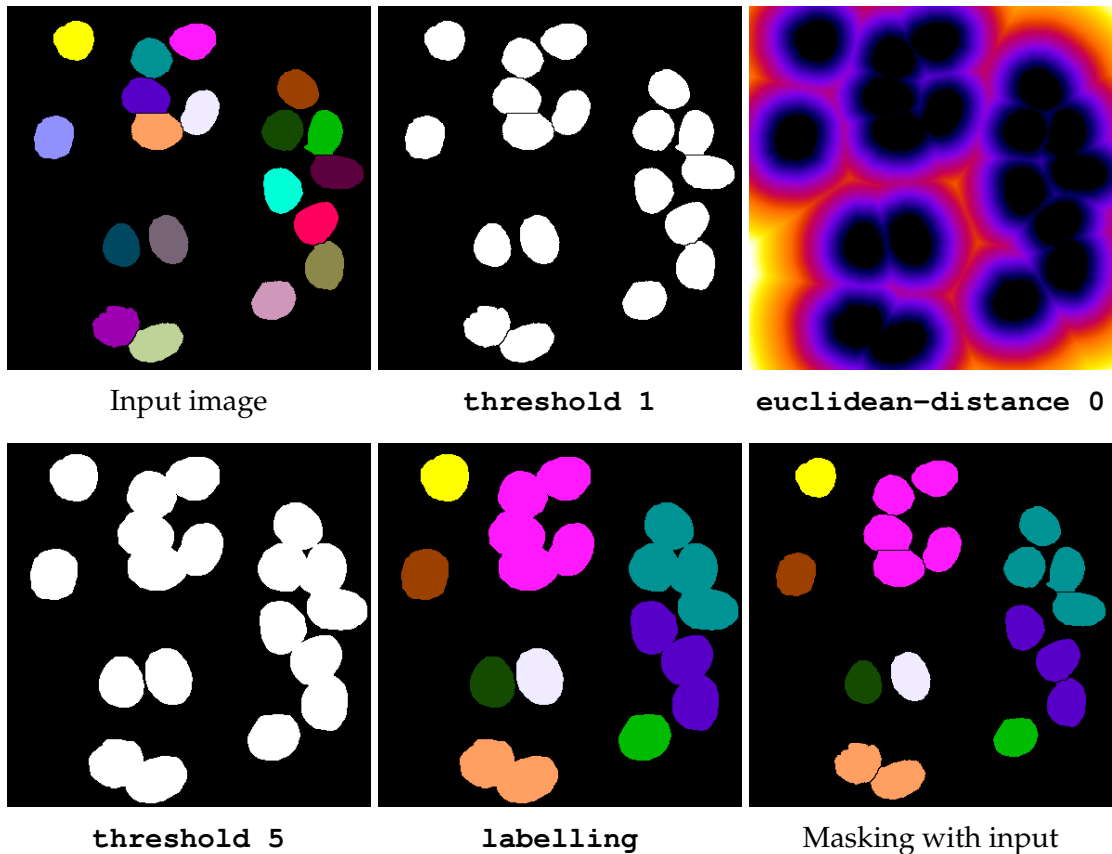


Figure 19: Labelling clusters of neighbouring objects (see [Section 13.4](#)).

13.5 Cell distance maps

Problem. Find the sequence of operators to compute the distance between each cell in a segmented tissue image to the innermost cells. Then write the corresponding pipeline.

Answer. Consider the innermost cells are the farthest ones from the cells at the periphery of the tissue. Therefore, the algorithm consists in first determining the innermost cells and then in computing the cell distance for each cell within the tissue to the inner core. This yields the following sequence of operations:

```
shell$ bip select-outer labels.tif
shell$ bip cell-distance labels-select-outer.tif labels.tif
shell$ bip select max labels-cell-distance.tif
shell$
    bip -f cell-distance labels-cell-distance-select.tif labels.tif
```

Note the **-f** option in the last call: as this is the second call to the operator **cell-distance** on image *labels.tif*, there would be a conflict on output filenames. The **-f** option enforces overwriting existing files.

The corresponding pipeline (illustrated in [Figure 20](#)) is:

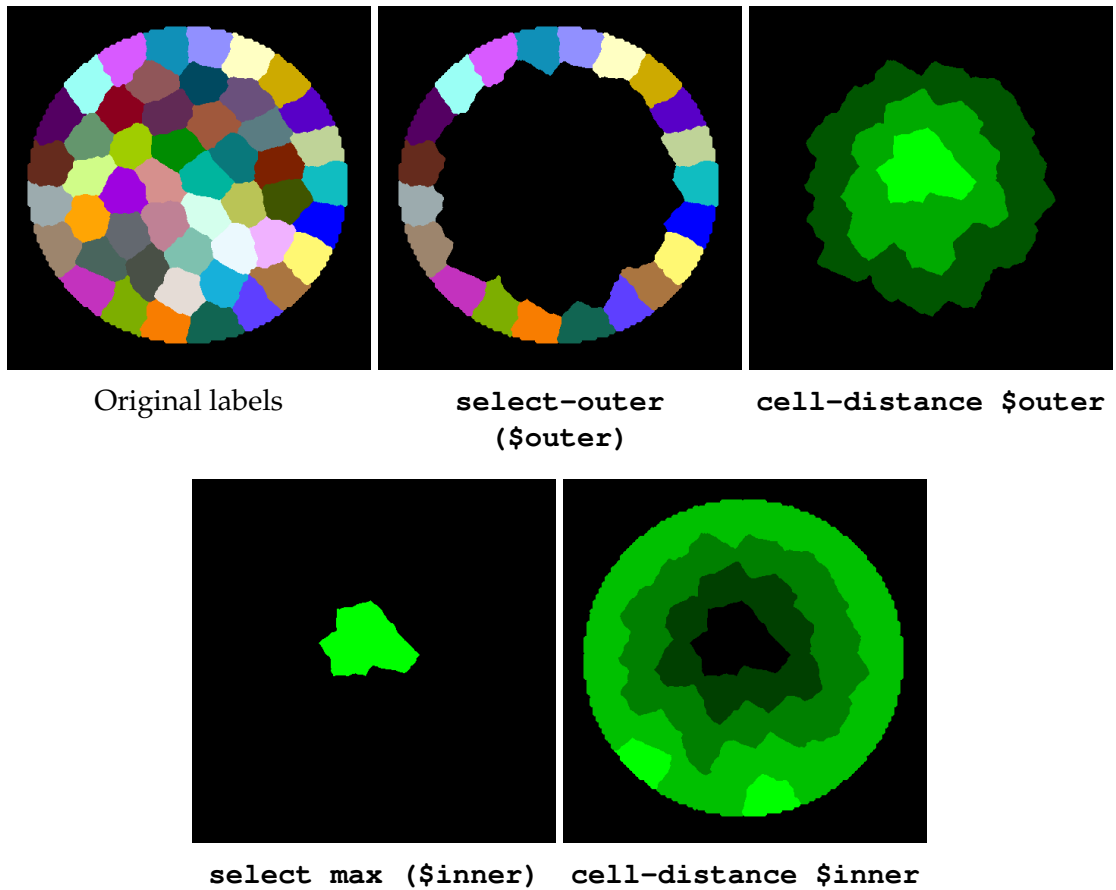


Figure 20: Computing cell distance from tissue center (see [Section 13.5](#)).

```

# make a copy of input labels
store $input

# select outer cells
select-outer
store $outer

# compute distance to outer cells
# and select the inner-most cells
recall $input
cell-distance $outer
select max
store $inner

# compute distance to inner cells
recall $input
cell-distance $inner

```

References

- Adobe Developers Association (1992). *TIFF™ 6.0 Specification*.
- Biot, E., Crowell, E., Höfte, H., Maurin, Y., Vernhettes, S., and Andrey, P. (2008). A new filter for spot extraction in N -dimensional biological imaging. In *Fifth IEEE International Symposium on Biomedical Imaging (ISBI'08): From Nano to Macro*, pages 975–978, Paris.
- Borgefors, G. (1986). Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344–371.
- Borgefors, G. (1996). On digital distance transforms in three dimensions. *Computer Vision and Image Understanding*, 64(3):368–376.
- Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698.
- Falk, T., Mai, D., Bensch, R., Çiçek, Ö., Abdulkadir, A., Marrakchi, Y., Böhm, A., Deubner, J., Jäckel, Z., Seiwald, K., Dovzhenko, A., Tietz, O., Dal Bosco, C., Walsh, S., Saltukoglu, D., Tay, T. L., Prinz, M., Palme, K., Simons, M., Diester, I., Brox, T., and Ronneberger, O. (2019). U-Net: deep learning for cell counting, detection, and morphometry. *Nature Methods*, 16:67–70.
- Felzenszwalb, P. F. and Huttenlocher, D. P. (2012). Distance transforms of sampled functions. *Theory of Computing*, 8(19):415–428.
- Keller, D., Stinus, S., Umlauf, D., Goubeyre, E., Biot, E., Olivier, N., Mahou, P., Beaupaire, E., Andrey, P., and Crabbe, L. (2024). Non-random spatial organization of telomeres varies during the cell cycle and requires lap2 and baf. *iScience*, 27:109343.
- Kramer, H. P. and Bruckner, J. B. (1975). Iterations of a non-linear transformation for enhancement of digital images. *Pattern Recognition*, 7(1–2):53–58.
- Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: a high resolution 3D surface construction algorithm. *ACM Computer Graphics*, 21(4):163–169.
- Meijster, A. and Wilkinson, M. H. (2002). A comparison of algorithms for connected set openings and closings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):484–494.
- Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66.
- Ridler, T. W. and Calvard, S. (1978). Picture thresholding using an iterative selection method. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(8):630–632.
- Rosin, P. L. (2001). Unimodal thresholding. *Pattern Recognition*, 34:2083–2096.
- Soille, P. (2003). *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, Berlin, Germany, second edition.

- Sternberg, S. R. (1986). Grayscale morphology. *Computer Vision, Graphics, and Image Processing*, 35:333–355.
- Vincent, L. (1993a). Grayscale area openings and closings, their efficient implementation and applications. In *Proceedings of the EURASIP Workshop on Mathematical Morphology and its Applications to Signal Processing*, pages 22–27, Barcelona, Spain.
- Vincent, L. (1993b). Morphological grayscale reconstruction in image analysis: applications and efficient algorithms. *IEEE Transactions on Image Processing*, 2(2):176–201.
- Welch, T. A. (1984). A technique for high-performance data compression. *Computer*, 17:8–19.
- Wolny, A., Cerrone, L., Vijayan, A., Tofanelli, R., Barro, A. V., Louveaux, M., Wenzl, C., Strauss, S., Wilson-Sánchez, D., Lymbouridou, R., Steigleder, S. S., Pape, C., Bailoni, A., Duran-Nebreda, S., Bassel, G. W., Lohmann, J. U., Tsiantis, M., Hamprecht, F. A., Schneitz, K., Maizel, A., and Kreshuk, A. (2020). Accurate and versatile 3D segmentation of plant tissues at cellular resolution. *eLife*, 9:e57613.
- Xue, J.-H. and Zhang, Y.-J. (2012). Ridler and Calvard’s, Kittler and Illingworth’s and Otsu’s methods for image thresholding. *Pattern Recognition Letters*, 33(6):793–797.
- Zack, G. W., Rogers, W. E., and Latt, S. A. (1977). Automatic measurement of sister chromatid exchange frequency. *The Journal of Histochemistry and Cytochemistry*, 25:741–753.