Textbook for 2nd Math-Clinic & Viroquant Bioimage Analysis Workshop

using Fiji and KNIME to solve:
2D spots counting,
2D+time tracking,
3D object-based colocalization

(Targeted for biologists with image analysis basics and starting image analysts)

Chong Zhang, Juergen Reymann

CellNetworks

University of Heidelberg, Germany

2014

CONTENTS

1

FIJI AND KNIME

1.1 Aim

The aim of this session is to familiarise you with ImageJ/Fiji and KNIME. This introduction session will be followed by three hands-on practical sessions on how to use these two tools for analysis of microscopy data and we will have a closer look at different Fiji functions and plugins, and several KNIME workflows.

1.2 Fiji

Fiji stands for “Fiji Is Just ImageJ". ImageJ is a public domain image processing and analysis program written in Java. It is freely available (`http://imagej.nih.gov/ij/index.html`) and used by many scientists all over the world. There is a very active ImageJ community and ImageJ is continuously improved. Fiji is a distribution of ImageJ together with Java, Java 3D and a lot of plugins organized into a coherent menu structure.

During this session we will have an introduction on the use of ImageJ/Fiji for digital image processing for life sciences, and also on how to write macros to analyze the data and how to reuse code. So during the practical sessions you will write code using the macro language but programming knowledge is not strictly required to follow the course.

Please refer to the homepage of Fiji for detailed documentation and user manual. Additional handouts for the material used in this course will be distributed during the session.

1.2.1 Installation and updates

After downloading Fiji installer, please follow the instructions from the Fiji page (`http://fiji.sc/Installation`) to install Fiji on your computer.

Fiji has an automatic update system invoked each time when you start Fiji, if there are newly available updates. You can also configure your own updates by calling [`Help -> Update Fiji`], and after checking for details, an `ImageJ Updater` window will appear with the list of items to be updated or installed (Fig. 1.1 *top*). If you would like to exclude/include specific items to be checked for update, or if you would like to add new sites that are not listed, you could click `Manage Update Site` to configure them (Fig. 1.1 *bottom*).

1.2.2 Plugins

Apart from the official plugins, there are many amazing plugins from a large community contribution. Normally you need to download a specific plugin from their website, and then copy it into the Fiji plugins folder. For example, for one plugin we will be using in this course, 3D ImageJ Suite (a package of multiple plugins): we can install it by downloading the bundle from `http://imagejdocu.tudor.lu/lib/exe/fetch.php?media=plugin:stacks:3d_ij_suite:mcib3d-suite.zip` and unzipping it in the plugins folder. And when we restart Fiji, the plugin should be found at [`Plugins -> 3D`].

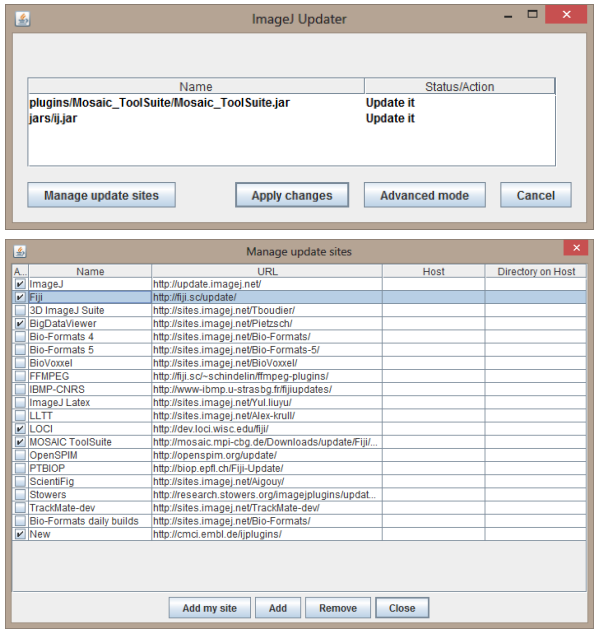


FIGURE 1.1: Fiji Updater.

1.3 KNIME: Konstanz Information Miner

KNIME features a user friendly graphical interface which enables basic users to create workflows for data analysis in a highly efficient manner. Each plugin is represented as a graphical node which can be configured individually and connected with other nodes.

KNIME in general is a large data mining platform including diverse libraries for data exploration like R (http://www.r-project.org/), Weka (http://www.cs.waikato.ac.nz/ml/weka/), clustering, network mining, machine learning and many others enabling you to process multi-dimensional data tables in a highly sophisticated manner. The basic idea is that - independent from the task - any kind of data is represented as data points within a data table. In case of life science applications, e.g. image analysis, the data points are indicated by features like intensity, size, shape, ... of objects.

KNIP (KNIME Image Processing) is a large image processing repository within KNIME which is based on the ImageJ2 (http://developer.imagej.net/) image processing library (ImgLib2: http://fiji.sc/wiki/index.php/ImgLib2). Functionalities as provided by ImageJ/Fiji are executable within KNIME and there exists a strong collaboration between KNIP and ImageJ/Fiji.

Due to the possibility to create workflows, KNIME is suitable for data pipelines of any kind incl. high-throughput data processing.

1.3.1 Installation

You can download KNIME from here: http://www.knime.org/downloads/overview. If you have problems with installation, we will install it during an installation session together. Additional repositories will be needed to be installed for running image processing applications.

- Execute the KNIME download file in a directory of your choice
- Start knime.exe

You will be asked about the workspace-directory (this is the directory, where your workflows will be saved by default). Choose a directory and enable 'Use this as the default and do not ask again'.

- In the next dialog window click `Open KNIME workbench`. This is the KNIME -desktop where you can configure and run your workflows
- Go to [`Help > Install New Software...`] and open the link "`Available Software Sites`". Enable the three software sites as depicted in the image below.
- Go to [`File > Install KNIME Extensions...`] and install the following plugins:
  1. KNIME Distance Matrix
  2. KNIME External Tool Support
  3. KNIME HTML/PDF Writer
  4. KNIME Math Expression (JEP)
  5. KNIME Nodes to create KNIME Quick Forms
  6. KNIME XLS Support
  7. KNIME XML-Processing
  8. KNIME HCS Tools
  9. KNIME Community Contributions - Imaging (entire package!)
  10. KNIME Decision Tree Ensembles
  11. KNIME Textprocessing
  12. KNIME Virtual Nodes
- ReStart knime.exe

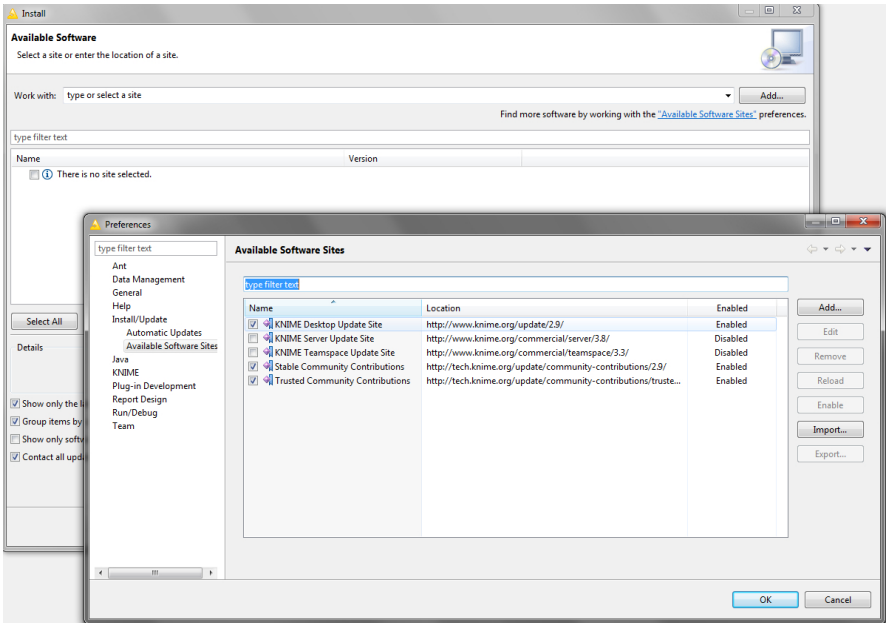


FIGURE 1.2: Snapshot of installation window: Available Software Sites.

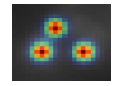
1.4 Resources

Fiji related:

- CMCI ImageJ Course page: http://cmci.embl.de/documents/ijcourses
- CMCI Macro programming in ImageJ Textbook: https://github.com/cmci/ij_textbook2/blob/master/CMCImacrocourse.pdf?raw=true
- Fluorescence image analysis introduction:http://blogs.qub.ac.uk/ccbg/fluorescence-image-analysis-intro/
- mailing list: imagej@list.nih.gov
- KNIP: KNIME Image Processing: http://tech.knime.org/community/image-processing

2

COUNTING SPOTS



2.1 Aim

In this session, we will count spots in image regions, with two different methods using Fiji and KNIME.

2.2 Introduction

In microscopy images, counting the number of objects is a rather common practice. When the density of objects in the image is low and the objects are well separated from each other, it is possible to count objects by first segmenting the objects and then perform a morphological operation called connected components analysis. However, as the density of the objects increases, such approach underestimates the number of objects due to under-segmentation of clustered objects.

Dataset In this session, we will use the example image, Hela cells, in the Session3 folder of the course material. You can also find it directly from Fiji sample data. This is a 16-bits/channel composite color image of Hela cells with red lysosomes, green mitochondria and blue nucleus. We will count the number of lysosomes in the whole image, find their distributions in e.g. cell nucleus or user-defined regions of arbitrary shape.

2.3 A Fiji solution

Before doing anything with Fiji, let's first turn on the later-to-be your favorite function ***ImageJ command recorder*** *(`Plugins -> Macros -> Record`). It magically records operation you will be doing with Fiji, which can be turned into a macro script either directly or with some modifications.*

If you would like to get the image directly from Fiji, then click on [`File->OpenSamples->HelaCells`]. Since we are interested in counting red lysosomes spots (Fig. 2.1), lets first split the channels using [`Image -> Color -> Split Channels`]. Or, just open the `C1-hela-cells.tif` from the session folder.

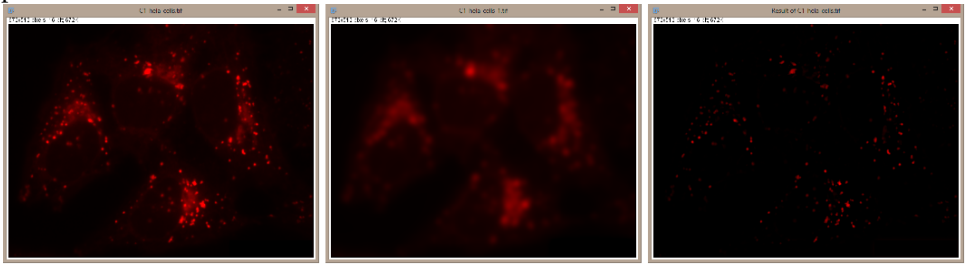


FIGURE 2.1: (*left*) The lysosome channel of the Hela cells image from Fiji sample data. (*middle*) estimated background using Gaussian smoothing. (*right*) The original image after subtracting the background estimate.

2.3.1 Detecting objects

2.3.1.1 Detecting and counting nucleus

Let's first select the nucleus channel image `C3-hela-cells.tif` (Fig. 2.2), since we are interested in the lysosomes inside the cell nucleus. This is a relatively "clean" image, to detect the nucleus we can try just thresholding and play with different methods and choose the best one, e.g. `Triangle`. Notice that the thresholded binary image is most likely not exactly what we want, but with small objects and possibly holes (Fig. 2.3). In such situations, a *morphological opening* operation can be used to clean up small objects. And for filling holes, a *morphological closing* operation can do the job. In this case, we need *morphological opening*, which is completed in two steps: a [`Process -> Filters -> Minimum`] followed by a [`Process -> Filters -> Maximum`]. In order to bring the object shape back to its original size, the `Radius` value for both filters should take the same value, e.g. 5.0 for this image. We can rename the image as "nucleus_mask".
Question: How the operations of *morphological closing* can be done?

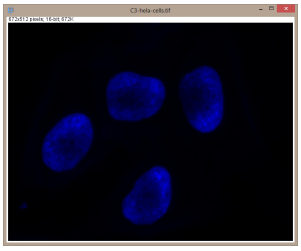


FIGURE 2.2: Nucleus channel

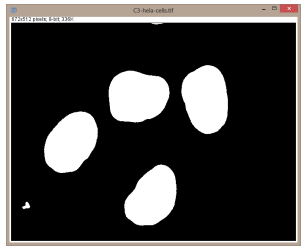


FIGURE 2.3: Thresholded

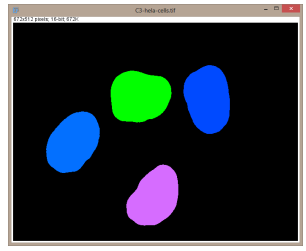
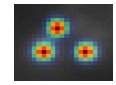


FIGURE 2.4: Cleaned mask

Now we can count the nucleus using [`Analyze -> Analyze Particles`], even though it is pretty easy to check by eye that there are 4 nucleus. We will set the options `Size`, `Circularity`, `Show` to: `0-Infinity`, `0-1`, and `Count Masks`, respectively, and verify that `Add to Manager` is unchecked while `In situ Show` is checked. Then we can click `OK`. In order to visualize the counted objects better, we could change the lookup table (LUT) option, using [`Image -> Lookup Tables`] options. If you like, we can also use the LUT file provided in your Session3 folder, "Random.lut". It should be copied to the ImageJ LUT folder. Once copied you should be able to find it in the `Lookup Tables` list and apply it (see Fig. 2.4). This is because when the option `Count Masks` is chosen, the mask image will result in an image with a different pixel value for each object. If we toggle the mouse cursor in the object regions, we will find their values, or object label or ID, displayed in the main Fiji menu. In this case, they should be 1-4 (and with zero background).

2.3.1.2 Detecting lysosome spots

In the image `C1-hela-cells.tif`, the blob-like highly-contrasted lysosomes spots exist in some cloud-shape background. So the first step is to estimate and subtract the



background. Since we need to subtract images, let's first make a copy of the image by running [Image -> Duplicate]. A typical approximation of such background is a much smoothed version of the image. So we could try the following choices and see which suits the best:

- run a *Gaussian smoothing* ([Process -> Filters -> Gaussian Blur]) with a large Sigma value (e.g. 6.00) on the duplicated image.

- apply a *morphological opening* to the image. This operation is done through a Minimum filter ([Process -> Filters -> Minimum]) followed by a Maximum filter ([Process -> Filters -> Maximum]) with the same Radius (e.g. 6.00). These filters have a Preview mode, which is helpful for choosing the satisfactory parameter(s) for the filters.

Once having a good approximation of the background, we can subtract it from the original image using [Process -> Image Calculator] with Subtract Operation. Alternatively, instead of first estimating and then subtracting background from the original image, we can also run [Process -> Filters -> Unsharp Mask] with a small Sigma value (e.g. 1.00) and large Mask Weight value (e.g. 0.9) directly. And we should be able to remove quite some cloud-like background signal and enhance signals of lysosomes.

Next, we will extract objects from the resultant "background-free" image. A first try can always be getting the binary mask after a thresholding. For example, we can try [Image -> Adjust -> Threshold] with the Otsu method or other ones. And then click Apply after we are satisfied with the thresholding to get a binary mask image. We could see that from this image (Fig. 2.5 *left*), some spatially close spots are connected and thus regarded as one object. So we would need to correct this undersegmentation behavior. One common solution is to run [Process -> Binary -> Watershed]. As we see in Fig. 2.5 *middle*, it separates some clustered spots (indicated in red circles) by a line with one-pixel width. It should be noted that the *watershed* method does not always do the magic, and it works better for objects with more regular shapes and sizes. After we managed to create a mask with one separate object for each spot, we could start identifying each object. First, let's run [Analyze -> Analyze Particles], After verifying whether Size, Circularity, Show options are set to: 0-Infinity, 0-1, and Nothing, respectively, and also Add to Manager is checked, then we can click OK. A new window similar to Fig. 2.5 *right* should appear. The numbers indicate the object (or ROI) ID. Let's rename this image as e.g. "spots" using [Image -> Rename]. And in the ROI Manager window we can find the list of all objects.

2.3.2 Counting spots

We will practice counting the spots that we just extracted from the lysosomes channel in two types of regions: nucleus and regions enclosed by mitochondria.

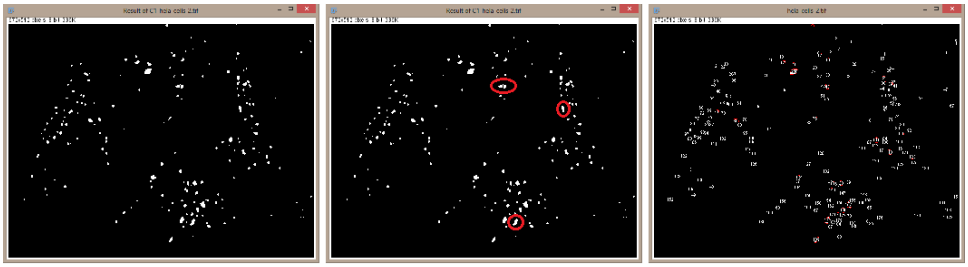


FIGURE 2.5: (*left*) The binary mask obtained from thresholding the background subtracted image. (*middle*) The binary mask after applying a watershed operation so as to separate obvious connected spots. (*right*) The labeled objects in the binary mask.

2.3.2.1 counting spots in nucleus

The trick to achieve this is to **apply the lysosome ROIs to the counted nucleus mask image, and then check the intensity of each ROI in this mask image, since the background and the four nucleus have different IDs.** So let's first go to [`Analyze -> Set Measurements`] and check only the `Min & max gray value` and uncheck the rest. Because we are interested in the maximum intensity value in each ROI.
Question: Why are we interested in the maximum intensity value in each ROI?
We need to select the "nucleus_mask" window, and highlight all the listed lysosome ROIs in the `ROI Manager` window. Then apply [`Image -> Overlay -> From ROI Manager`]. Your nucleus mask image should look like Fig. 2.6.

Then we can measure the parameters we just set, by clicking `Measure` in the `ROI Manager` window. The `Results` window is shown with three columns: object/ROI ID, minimum intensity and maximum intensity. Now if we click [`Results -> Distribution`] in the same `Results` window, and select `Max` as `Parameter`, and set `specify bins` and `range` to 5 and 0-5, respectively, a `Max Distribution` window should appear (see Fig. 2.7) with five bins/bars, whose heights indicate the number of ROIs with their maximum intensity value between the range of each histogram bin. *As we know that the mask has intensity range 0-4, if the histogram bins are chosen such that each bin represents the intensity range for each nucleus and the background (i.e. 1 intensity level per histogram bin in this case), the histogram does the counting for us.* That's the reason for specified values for bins and range.

In order to see the exact number of each counts, we can click `List` in the histogram window, and a two-column table similar to Fig. 2.7 will appear with the intensity `value` column indicating the nucleus ID, and the `count` column showing the number of ROIs.

2.3.2.2 counting spots in regions where mitochondria also presents

Similarly, if we are interested in counting spots in the regions covered by mitochondria, we can perform similar steps but on the mitochondria channel (`C2-hela-cells.tif`).

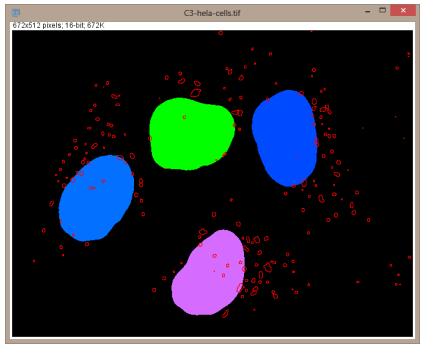


FIGURE 2.6: Nucleus mask & spot ROIs.

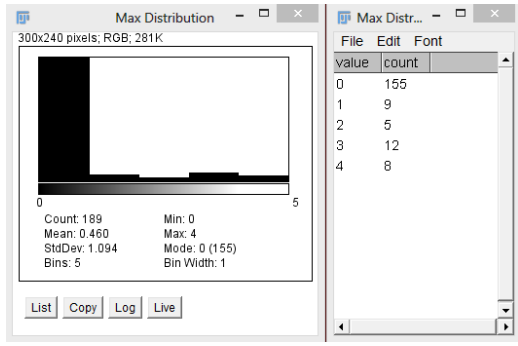


FIGURE 2.7: Histogram & counts in nucleus.

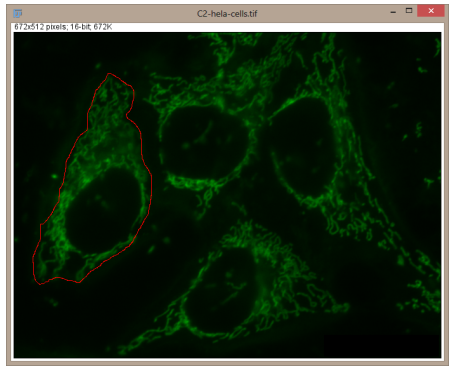


FIGURE 2.8: The mitochondria channel with manual selected region of interest.

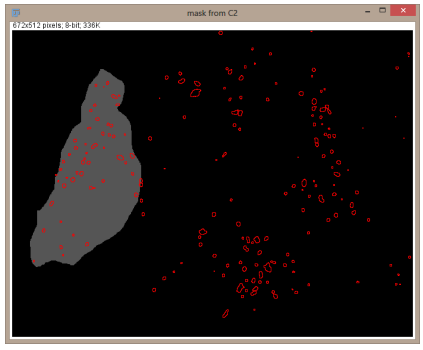


FIGURE 2.9: Nucleus mask & spot ROIs.

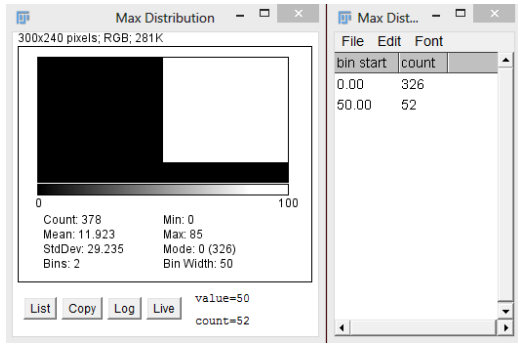


FIGURE 2.10: Histogram & counts in mitochondria.

Instead of using some filters to extract such regions, we will practice with manually drawing a contour of the region of interest using the `Freehand selections` tool (see Fig. 2.8). In previous steps, the counting was done through computing histogram from the labeled mask image. Similarly, a mask image is needed here. So we will create a new mask image from the manual selection. There are many options to do so, and a few of them are listed below:

- A new image of the same size is created using [`File -> New -> Image`] and renamed as “mito_mask". It will create a black image, and when we run [`Edit->Selection->RestoreSelection`], the manually drawn region contour on the mitochondria channel will be “translated" to the new image. Then the pixel values inside this region should be modified to distinguish it from the dark background, using:
  - the command [`Edit -> Fill`]. The filled region will have a value higher than the zero background, e.g. 85 (see this image in Fig. 2.9).
  - the command [`Process -> Math -> Add`], and we can decide which value to add to the new image for the selected region.
- After manually drawn region, we run [`Edit->Selection->CreateMask`], a new binary image with the same dimension is created, with the manually drawn region highlighted (i.e. 255) and the rest being 0.

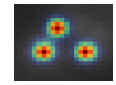
The next steps are similar to what we have done in the previous section thus please refer to it for detailed descriptions. One example result is shown in Fig. 2.10. Please note that since there are only two values in the image (0 and 85), then we could have many histogram bin distribution options, as long as 0 and 85 are in separate bins.

In case of multiple regions to be measured, we could hold the “shift" key while drawing multiple regions, and then run [`Edit->Selection->CreateMask`]. Or use [`Process -> Math -> Add`] each time after drawing one region. If different values are added for each region, then a labeled mask image is created directly.

2.3.3 Convert the recorded commands into a macro script

We will clean up the commands recorded from the beginning so as it becomes a reusable macro script for other datasets. Actually if it makes things easier, you could clean it parts by parts throughout the recording period, which we will be doing so during the session. Cleaning up mostly involves: deleting mistaken commands, specifying selections like selected windows or images, replacing certain parameter settings by variables with values specified once manually or through programming. We will gradually practice this during the whole course.

The macro code obtained and cleaned up from the ImageJ command recorder (`Plugins -> Macros -> Record`) is provided in the Session3 folder.



2.4 KNIME solution

This session is divided in two parts: The first one [Session3a_SpotsInNuclei] detects nuclei and spots inside the nuclei without taking care of surrounding regions, i.e. mitochondria. The second one [Session3b_SpotsInNucleiSurroundings] provides the full solution including spots in mitochondria.

Nomenclature: In the following, the expression `node-name || config1 | config2 | ...` defines a KNIME -node and the corresponding configuration. Please note, that only configurations which differ from the default node configuration are mentioned. Apart from that the full node details will not be described. For this please have a look at the node description of the KNIME software.

2.4.1 Spots in nuclei

Open the workflow [Session3a_SpotsInNuclei].

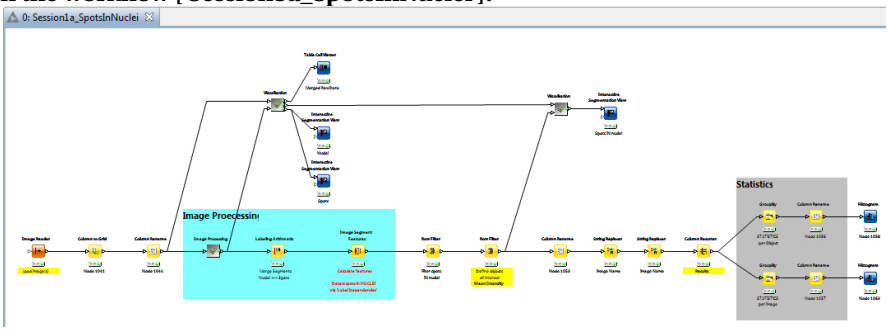


FIGURE 2.11: KNIME workflow *Session3a_SpotsInNuclei*.

After injecting the images into the workflow via the `Image Reader` node, the `Column to Grid || Grid Column Count: 2` node transfers the table from a [1 column / 2 rows] into a [2 columns / 1 row] table. This procedure enables to rename the two images column wise into *Spots Channel* and *Nuclei Channel*.

Within the Image Processing MetaNode we use two different branches and methods resp. to segment the images:

- Nuclei as performed in the INTRO session
- Spots are detected using the node `Spot Detection || enable wavelet level 0...:2 | disable wavelet level 1... | disable wavelet level 2...` .

We then combine the results of this first image processing step using the `Joiner` node. The result looks as follows (Fig. 2.13):

Since we are interested in spots within the nuclei we can merge the two labeled images *Compose Labeling(Bitmask)* (nuclei) and *Spots Channel_thresh* (spots) using the

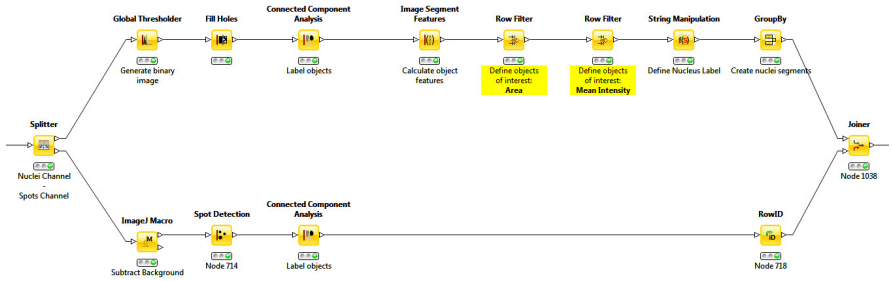


FIGURE 2.12: Content of the Image Procesing MetaNode: The upper branch segments the nuclei. The lower branch calculates spots segments.

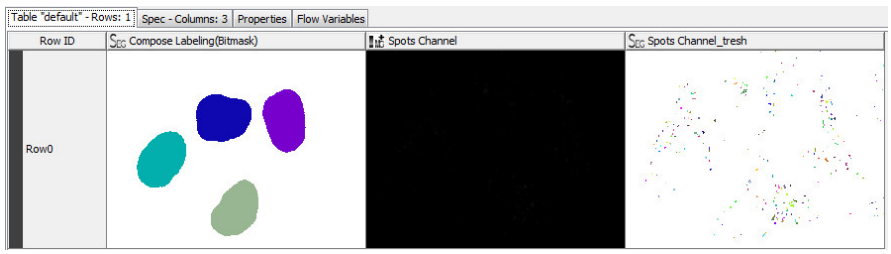


FIGURE 2.13: Result of the Image Processing MetaNode.

`Labeling Arithmetic || Method: Merge` node. Resulting we obtain an overlay between the segmented spots and nuclei (Fig. 2.14):

Using a new functionality in the `Image Segment Features || Segment Settings: enable *Append labels of overlapping segments*` node allows us to extract information about objects which overlap. In our case these are the spots within the nuclei. The information is listed in the results table of the `Image Segment Features` node in the *LabelDependencies* column. I.e. that a segment (=spot) with Label xxx in the first image overlaps with a segment (=nucleus) of the second image in case of a positive *LabelDependency* entry.

The following `Row Filter` node filters out the positive matches of *LabelDependencies* providing the result as follows (after executing the visualization steps) (Fig. 2.17):

After that we perform some basic statistics and obtain the overall results from the image processing (Fig. 2.18):

The table lists the *Image Name*, the *spot labels*, the spot coordinates *[X]* and *[Y]*, the spot *Area*, *Max* and *Mean* spot intensity and the *LabelDependencies* enabling to assign each spot to the corresponding nucleus. In addition, the table contains the *Bitmasks* and *Source Labeling* of each of the spots in case that you are interested

FIGURE 2.14: Result of the `Labeling Arithmetic` node: Overlay of the nuclei and spots segments.

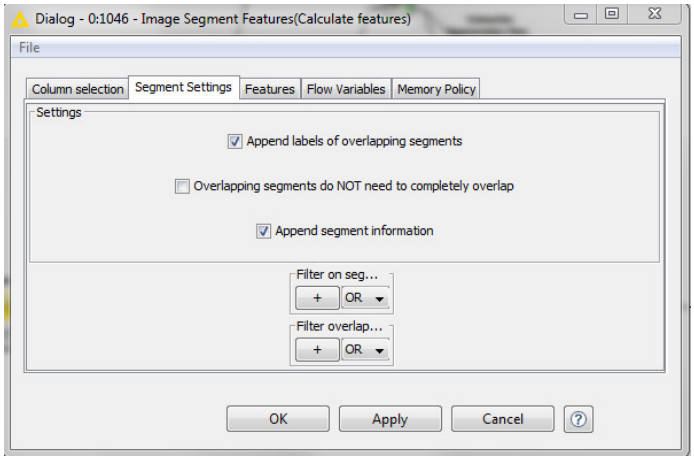


FIGURE 2.15: Configuration dialog of the `Image Segment Features` node. The *Append lables of overlapping segments* flag provides information if a spot is located in a nucleus.

in visualizations.

Within the last step we perform some basic statistics *per object* / *per image* using the `GroupBy || ...` node in order to obtain final numbers (Fig. 2.19):

For a graphical representation of the results we can e.g. use the `Histogram` nodes (Fig. 2.20):

File

Table "default" - Rows: 332 | Spec - Columns: 9 | Properties | Flow Variables

Row ID	Bitmask	Label	Source Labeling	LabelDependencies	Centroi...	Centroi...	Num Pix	Max	Mean
Row0#35		35	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		320.667	84.667	3	352	339.333
Row0#36		36	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		442	87.2	10	855	660.4
Row0#33		33	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		158.333	86.667	15	942	743.533
Row0#159		159	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		661	222.2	5	389	337.4
Row0#34		34	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		309.579	87.658	38	1,845	1,316.658
Row0#158		158	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		96	221	1	189	189
Row0#39		39	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		300.25	90	4	368	313
Row0#157		157	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		543	219	1	628	628
Row0#156		156	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...	Nucleus_4	189	220.231	13	858	745.923
Row0#37		37	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		138.917	90.167	12	813	667.167
Row0#155		155	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		547.867	219.533	15	2,395	1,772.2
Row0#38		38	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		333	88	1	302	302
Row0#154		154	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...	Nucleus_4	189	218	1	610	610
Row0#152		152	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		117.167	219.833	18	1,749	1,365.833
Row0#153		153	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		126.533	220.133	15	2,006	1,493.533
Row0#150		150	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...	Nucleus_3	341.8	215.5	10	691	549.7
Row0#151		151	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		84.267	218.333	15	1,174	848.6
Row0#43		43	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		378	93	1	310	310
Row0#42		42	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		151.5	93	6	551	469
Row0#41		41	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		365	91.5	2	262	256.5
Row0#40		40	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		334	89	1	319	319
Row0#202		202	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		532.5	277	2	315	305.5
Row0#203		203	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		355.5	280	6	406	341.333
Row0#204		204	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		552.429	281.286	7	665	554.429
Row0#205		205	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		215.267	283.333	15	1,621	1,151.267
Row0#200		200	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		517.93	280.744	43	2,863	2,057.442
Row0#201		201	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...		580.333	275.667	3	414	350.333

FIGURE 2.16: Result table of the `Image Segment Features` node: The column *LabelDependencies* provides the information if a spot is located in a nucleus (of label x).

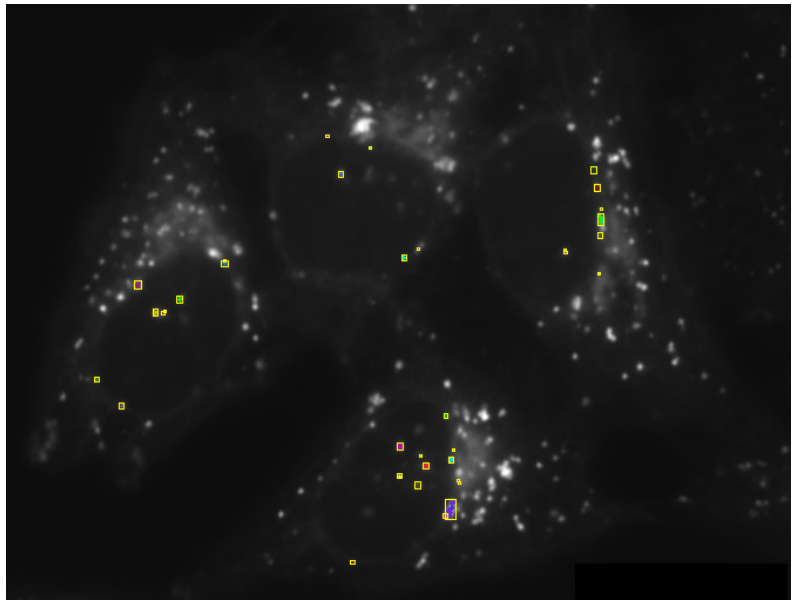
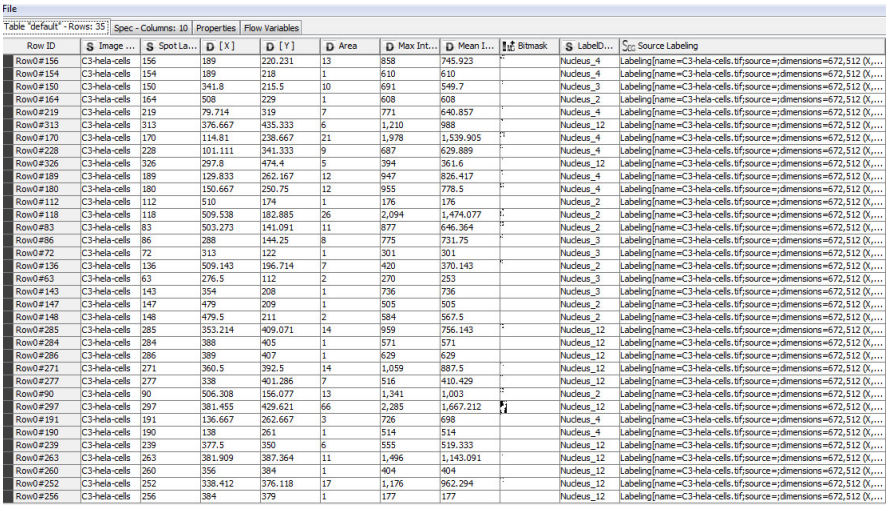


FIGURE 2.17: Outcome of the visualization steps: Overlay of image raw data and identified spots within the nuclei.

Now we can proceed and extend the workflow towards the detection of spots within mitochondria, i.e. nuclei surroundings.

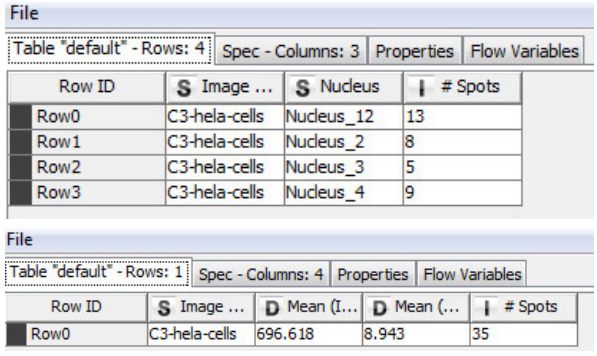


File

Table "default" - Rows: 35 | Spec - Columns: 10 | Properties | Flow Variables

Row ID	Image ...	Spot La...	[X]	[Y]	Area	Max Int...	Mean I...	Bitmask	LabelD...	Source Labeling
Row0#156	C3-hela-cells	156	189	220.231	13	858	745.923		Nucleus_4	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#154	C3-hela-cells	154	189	218	1	610	610		Nucleus_4	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#150	C3-hela-cells	150	341.8	215.5	10	691	549.7		Nucleus_3	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#164	C3-hela-cells	164	508	229	1	608	608		Nucleus_2	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#219	C3-hela-cells	219	79.714	319	7	771	640.857		Nucleus_4	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#313	C3-hela-cells	313	376.667	435.333	6	1,210	988		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#170	C3-hela-cells	170	114.81	238.667	21	1,978	1,539.905		Nucleus_4	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#228	C3-hela-cells	228	101.111	341.333	9	687	629.889		Nucleus_4	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#326	C3-hela-cells	326	297.8	474.4	5	394	361.6		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#189	C3-hela-cells	189	129.833	262.167	12	947	826.417		Nucleus_4	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#180	C3-hela-cells	180	150.667	250.75	12	955	778.5		Nucleus_4	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#112	C3-hela-cells	112	510	174	1	176	176		Nucleus_2	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#118	C3-hela-cells	118	509.538	182.885	26	2,094	1,474.077		Nucleus_2	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#83	C3-hela-cells	83	503.273	141.091	11	877	646.364		Nucleus_2	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#86	C3-hela-cells	86	288	144.25	8	775	731.75		Nucleus_3	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#72	C3-hela-cells	72	313	122	1	301	301		Nucleus_3	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#136	C3-hela-cells	136	509.143	196.714	7	420	370.143		Nucleus_2	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#63	C3-hela-cells	63	276.5	112	2	270	253		Nucleus_3	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#143	C3-hela-cells	143	354	208	1	736	736		Nucleus_3	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#147	C3-hela-cells	147	479	209	1	505	505		Nucleus_2	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#148	C3-hela-cells	148	479.5	211	2	584	567.5		Nucleus_2	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#285	C3-hela-cells	285	353.214	409.071	14	959	756.143		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#284	C3-hela-cells	284	388	405	1	571	571		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#286	C3-hela-cells	286	389	407	1	629	629		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#271	C3-hela-cells	271	360.5	392.5	14	1,059	887.5		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#277	C3-hela-cells	277	338	401.286	7	516	410.429		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#90	C3-hela-cells	90	506.308	156.077	13	1,341	1,003		Nucleus_2	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#297	C3-hela-cells	297	381.455	429.621	66	2,285	1,667.212		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#191	C3-hela-cells	191	136.667	262.667	3	726	698		Nucleus_4	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#190	C3-hela-cells	190	138	261	1	514	514		Nucleus_4	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#239	C3-hela-cells	239	377.5	350	6	555	519.333		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#263	C3-hela-cells	263	381.909	387.364	11	1,496	1,143.091		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#260	C3-hela-cells	260	356	384	1	404	404		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#252	C3-hela-cells	252	338.412	376.118	17	1,176	962.294		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...
Row0#256	C3-hela-cells	256	384	379	1	177	177		Nucleus_12	Labeling[name=C3-hela-cells.tif;source=;dimensions=672,512 (X,...

FIGURE 2.18: Result of the `Column Resorter` node.



File

Table "default" - Rows: 4 | Spec - Columns: 3 | Properties | Flow Variables

Row ID	Image ...	Nucleus	# Spots
Row0	C3-hela-cells	Nucleus_12	13
Row1	C3-hela-cells	Nucleus_2	8
Row2	C3-hela-cells	Nucleus_3	5
Row3	C3-hela-cells	Nucleus_4	9

File

Table "default" - Rows: 1 | Spec - Columns: 4 | Properties | Flow Variables

Row ID	Image ...	Mean (I...	Mean (...	# Spots
Row0	C3-hela-cells	696.618	8.943	35

FIGURE 2.19: Overall results.

2.4.2 Spots in nuclei surroundings

Open the workflow [Session3b_SpotsInNucleiSurroundings].

The only difference is given by an additional branch in the Image Processing MetaNode which is defined by the `Voronoi Segmentation` node (Fig. 2.21):

The `Voronoi Segmentation || ...` node is normally used to segment cytoplasm regions based on certain intensity levels.

However, we can use this node to detect the mitochondria areas. As input the node requires for a mask which acts as the region where surroundings have to be detected

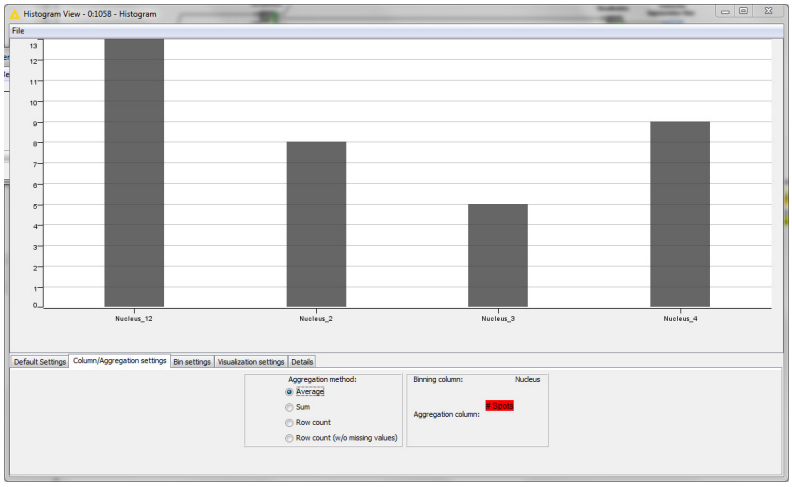


FIGURE 2.20: Histogram of the outcome of the upper `GroupBy` node. Shown is the number of spots per nucleus.

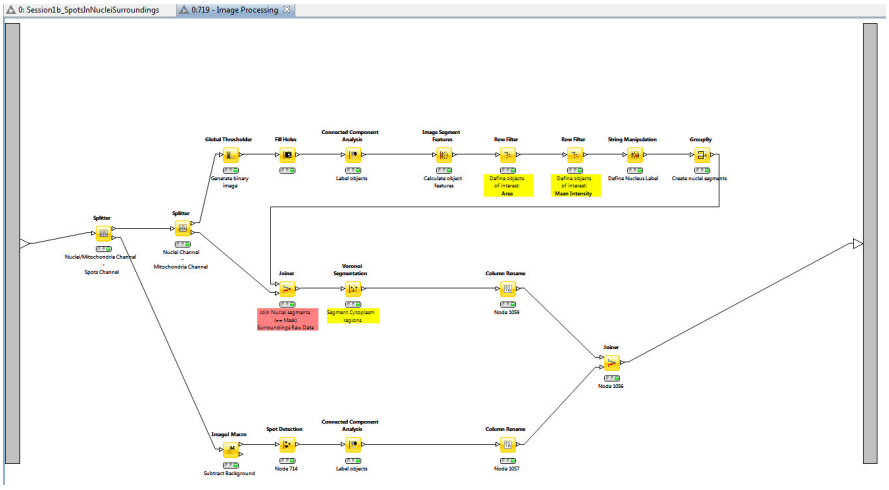


FIGURE 2.21: The modified Image Processing MetaNode. The additional branch (including the `Voronoi Segmentation` node) segments the nuclei surroundings based on the mitochondria intensity.

(*Seed regions* in the configuration dialog). In our case this mask is given by the nucleus segmentation. The corresponding surrounding (Voronoi-) regions are calculated by using

the image raw data of the mitochondria channel which contains the information of the pixel intensity values (Fig. 2.22):

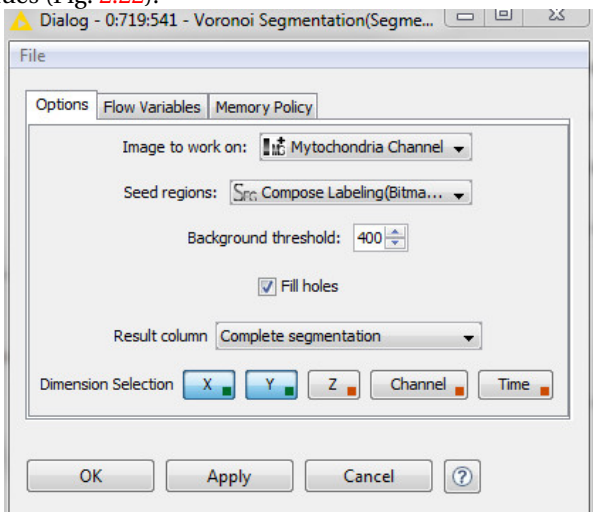


FIGURE 2.22: Configuration dialog of the `Voronoi Segmentation` node.

By changing the `Voronoi Segmentation || Background threshold ...` values the surrounding region of interest can be defined based on intensity values of the image raw data. The result of the `Voronoi Segmentation` node looks as follows with the surrounding regions/segments in the last column (Fig. 2.23):

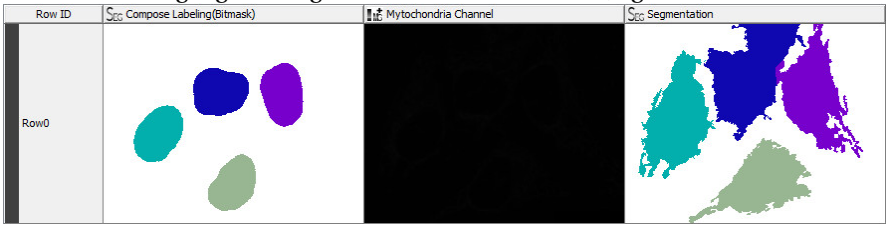


FIGURE 2.23: Result of the `Voronoio Segmentation` node: The Voronoi regions, i.e. nuclei surroundings are depicted in the last column.

Accordingly to session 1a, we can run the following nodes in order to finalize the story. Please note that the label *Nucleus_x* in the *LabelDependencies* column now stands for the entire Voronoi region, i.e. surroundings plus corresponding nucleus!

2.5 Tips and tools

- Search for commands in Fiji: select Fiji general interface, then hit the “L" key (or sometimes in MAC OS “`command`+L" key)

- ImageJ command recorder: `Plugins -> Macros -> Record`

2.6 Groups close by which work on similar problems

- Prof. Fred Hamprecht's group, Multidimensional Image Processing Group (http://hci.iwr.uni-heidelberg.de/MIP/), located at HCI, Speyererstr. 6

3

2D+TIME TRACKING

3.1 Aim

In this session, we will perform 2D+time tracking on objects of interest with two different methods: Fiji and KNIME. And further tracking results analysis of plotting trajectory on image and drawing displacement or velocity plot will also be done.

3.2 Introduction

Time-lapse experiments play a crucial role in current biomedical research on e.g. signaling pathways, drug discovery and developmental biology. Such experiments yield a very large number of images and objects such as cells, thus reliable cell tracking in time-lapse microscopic image sequences is an important prerequisite for further quantitative analysis.

Dataset The dataset we will use ("mitocheck_small.tif") is kindly provided by the publicly available database of the MitoCheck project (`http://www.mitocheck.org/`). One of the focuses of the MitoCheck project is on accurate detection of mitosis (cell division). Please refer to [1] for more details of this project.

3.3 A Fiji solution

Let's load the "mitocheck_small.tif" 2D+time image. The very first step is to segment, or identify, objects (cell nucleus in this case) in each time frames. This step can be done by first smoothing a bit the stack to homogenize a bit the intensity within each object using [`Process -> Filters -> Gaussian Blur`] by a Sigma value of e.g. 2.0. Please note that although we are smoothing the whole stack, this 2D filter applies to each slice, or time frame, independently. Since the background is quite clean, a simple thresholding is probably good enough to segment the objects. You can choose your favorite thresholding method and set the best threshold value, e.g. with the Default or the Moments method, and with min and max threshold value set to 15 and 255, respectively. Probably there will be merged objects, so we can run [`Process -> Binary -> Watershed`] to split the most obviously wrongly-merged ones. We could also run [`Process -> Filters -> Minimum`] with Sigma of 1.0 to shrink a little bit the obtained binary mask. This is to avoid merging of close neighboring objects in 3D in further steps, and also to correct possible dilation of objects due to the Gaussian blur filter. If we wish, a second Watershed operation can be applied afterwards to further split potential wrong merges.

Suppose that we have so far obtained a binary stack containing the segmented cell nucleus. Next, we will track, or link, the same object in consecutive time frames. The final result will be a label stack where each object is identified (filled) by a unique label along time (indicated by a distinct gray level intensity value). **The strategy we employ here is based on the overlap of the same object in temporal space between two consecutive time frames. If we consider time as the third dimension, cell spatial overlap along time translates to the connected parts in the third dimension of a 3D object.** And Fiji's [`Analyze -> 3D Objects Counter`] does the job of finding such connected

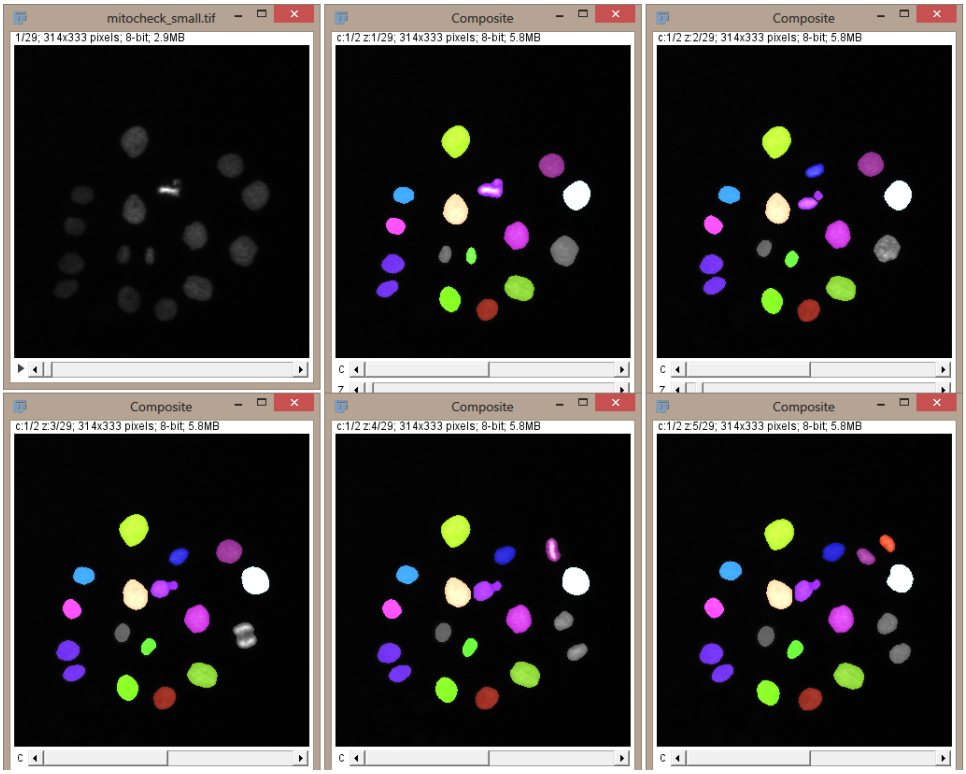


FIGURE 3.1: An example MitoCheck image (first frame) and the first five frames of the tracked objects (in different colors), using our Fiji solution, overlaid on the original image stack.

components in 3D. This function works similarly as the 2D one we are familiar by now - [`Analyze -> Analyze Particles`]. It can identify and count 3D objects in a stack, quantify each object's properties (see the full list in [`Analyze -> 3D OC Options`]) in a table, and generate maps of specified results representations. Once the 3D objects are 3D labeled, we can apply the `Random` LUT in the [`Image -> Lookup Tables`] to visualize better individual objects. We could also merge the label stack with the original stack to check results, using [`Image -> Color -> Merge Channels`]. You can specify the two stacks in two of the available channels. And by unchecking the option `"Ignore source LUT"`, it allows keeping the random LUT in the label channel thus colored IDs after merging. In Fig. 3.1 the first five frames of the objects (with their identity labels shown in random color) are shown.

Please note that the method we use here works properly only in cases: where single objects has spatial overlap in temporal space between any two consecutive time frames; and also a single object at a later time point does not overlap multiple objects at an ear-

lier time point. If there is no overlap, or connection between two consecutive frames, then a new label will be assigned to the same object in the latter frame, since it is considered as another object just showing up. One example can be found in Fig. 3.1. The two objects colored in orange and purple at the upper right corner of the last frame are split from the purple one in the previous frame. So both should haven been assigned the same identify (i.e. purple), but since the orange one has no overlap with its previous time frame, a new identity is assigned instead.

3.3.1 Other possibilities to solve tracking problem

An alternative to the 3D Object Counter could be `Plugins -> Process -> Find Connected Regions`. It sometimes could be faster than the 3D Object Counter and sometimes has better options (like starting from a point selection), but also lacks some flexibility.

There are other advanced tracking tools (included in Fiji or downloadable as Fiji Plugins) available such as [`Plugins -> Mosaic -> Particle Tracker 2D/3D`][1] and [`Plugins -> Tracking -> TrackMate`]. These trackers are however optimized for spot-like particle tracking, the linking is hence performed over a list of spot positions (spots detected in each frame). The tracking can be either straightforward (e.g. linking a spot to the closest spot in the next frame), or with algorithms that can handle cases when splitting and merging events occur.

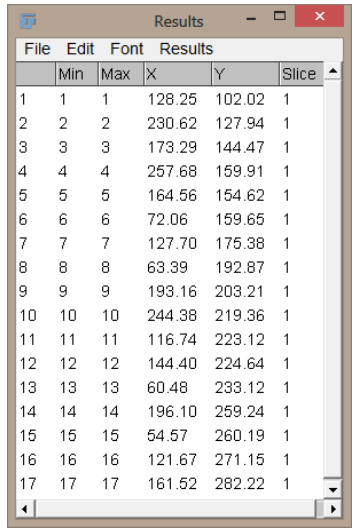
If the cells, or objects of interest, resemble ellipsoids and are sufficiently separated these trackers might perform well provided the parameters are properly adjusted. For a densely packed scenario, or for more irregular object shapes, a possible strategy is to firstly perform a specific segmentation and generate a map showing for instance the objects centroids, and then apply the spot tracker on the centroids maps. For those who are interested in, an example on multicellular movement in Drosophila with detailed explanation can be found in the BioImage Data Analysis Course at EuBIAS 2013 (http://eubias2013.irbbarcelona.org/bias-2-course).

3.4 Tracking analysis in Fiji: plot trajectory, displacement, or velocity

Once the objects are correctly identified and tracked along time, we could do some further analysis such as extracting trajectories (Fig. 3.3) and calculating displacements and velocities (Fig. 3.4). We will now find out how to do this in Fiji. In this practice, we will write our own Macro script. By now, we are at ease with the great function `Plugins -> Macros -> Record`. Next we will try to take advantage of some ImageJ built-in functions. Please note that the plotting functionality in Fiji is probably not amongst the most powerful and versatile ones, but it still can manage basic demands. We would like to introduce it as a quick checking means before going for more complex scenes.

Let's treat each object as a rigid body, meaning that everywhere inside the object has exactly the same dynamic behavior. Therefore, we could simplify the problem and look at just one specific point of the object. Good candidates could be the centroid and center of

[1] download site: http://mosaic.mpi-cbg.de/Downloads/Mosaic_ToolSuite.jar



	Min	Max	X	Y	Slice
1	1	1	128.25	102.02	1
2	2	2	230.62	127.94	1
3	3	3	173.29	144.47	1
4	4	4	257.68	159.91	1
5	5	5	164.56	154.62	1
6	6	6	72.06	159.65	1
7	7	7	127.70	175.38	1
8	8	8	63.39	192.87	1
9	9	9	193.16	203.21	1
10	10	10	244.38	219.36	1
11	11	11	116.74	223.12	1
12	12	12	144.40	224.64	1
13	13	13	60.48	233.12	1
14	14	14	196.10	259.24	1
15	15	15	54.57	260.19	1
16	16	16	121.67	271.15	1
17	17	17	161.52	282.22	1

FIGURE 3.2: Example `Results` window of the first slice/frame showing the requested measurements values from all objects in this slice.

mass. Let's take the centroid as example for illustration. The task now is to obtain a list of the centroid coordinates in every slice where the object of interest is present. This means that each slice of the original stack should be processed individually. Beware to process the correct slice/time and record it to the correct time indices of centroid coordinates. To this purpose, use the built-in macro function **setSlice()**. A **for** loop could be implemented to go through all the slices, using **nSlices**, a built-in macro function returning the value of the number of slices (or frames) of the active stack. In order to obtain object centroid coordinates, we could use the `Analyze Particles` command to extract objects and check the `Centroid` item in the `Analyze -> Set Measurements` options window. Additionally, in order to know which extracted centroids is the current object we are interested in, we should also check `Min & max gray value` in the same `Analyze -> Set Measurements` options window. Since in previous steps we have identified the objects over time through assigning each object a unique intensity value as its label ID, then by checking the min or max gray value of this label image tells us which object centroid we should be look for in each slice. When we run `Analyze -> Measure` or click `Measure` directly in the `ROI Manager` window, the checked measurements items will be displayed in a table shown in the `Results` window (Fig. 3.2). In order to retrieve the table information, the built-in functions **getResult** and **nResults** can help, where **getResult("Column", row)** returns a measurement from the `Results` table of in total **nResults** rows. For example in Fig. 3.2, **getResult("X", 16)** will return value 161.52, which is the value of the last but also the **nResults**th row (where **nResults**=17 but it is indexed as 16 since the first row has index of 0). Once we identify an object with its ID specified by the

intensity value in the label image, we can go through the objects list in the current slice and look for it, using an **if** sentence to check this condition by comparing the ID values - "**Max**" value (in this case "**Min**" or "**Mean**" values are also fine since they have the same value). And once the condition is met, the centroid coordinates can be obtained by getting the Columns "**X**" and "**Y**".

The following piece of the source code shows the corresponding part that implements the steps described above. The mentioned built-in functions are highlighted in brown color. Please note that we clean up the `Results` table and the `ROI Manager` after finishing checking each slice.

```
//The object of interest, specified by "objID"
objID = 16;

//Define and initialize an array with a time flag for each frame, whether
    the object "objID" is shown
objShowTime = newArray(nSlices);
Array.fill(objShowTime,0);
//Define and initialize two arrays to store the centroid coordinates in X
    and Y axes
objCenterX = newArray(nSlices);
Array.fill(objCenterX,-1);
objCenterY = newArray(nSlices);
Array.fill(objCenterY,-1);

//Check the measurements options, i.e. min & max gray value and centroid
run("Set Measurements...", "  min centroid redirect=None decimal=2");

for(i=1;i<=nSlices;i++)
{
  //Select the binary mask stack image, with image ID "cpID", and get the
    slice i
  selectImage(cpID);
  setSlice(i);
  //analyze the regions of each object in slice i, and add the regions to
    roiManager, to obtain ROI selection
  run("Analyze Particles...", "size=0-Infinity circularity=0.00-1.00 show=
    Nothing add");

  //We would like to measure the min/max gray value in the object label
    image so as to get object "objID"
  selectImage(lblID);
  setSlice(i);
  roiManager("Measure");

  //Going through all the shown objects in the current slice to look for the
     object with "objID"
  for(j=0;j<nResults;j++)
  {
    current_objID=getResult("Max", j);
    if (current_objID==objID)
    {
      print(current_objID);
      objShowTime[i-1]=1;
      objCenterX[i-1]=getResult("X", j);
```

```
      objCenterY[i-1]=getResult("Y", j);
    }
  }

  //Clean up, for each slice, the Results table and also the ROI Manager
  run("Clear Results");
  selectWindow("ROI Manager");
  run("Close");
}
```

:

3.4.1 Plot trajectories on image

In this exercise, we aim at practicing ImageJ built-in functions to extract measurements from the `Results` table and further plot object trajectory on image. For illustration, the simplest situation of plotting one trajectory from a single object at a time is considered.

In Fiji, we could use the built-in function **makeLine** in order to plot the trajectory of the centroid positions overlaid on the image (e.g. the first frame or the first frame when the object shows up). Since this function only takes integer pixel positions, the function **floor** is used to always take the integer part of the coordinates and omit the decimal part. The code below realizes the trajectory we want to plot. It should be inside a **for** loop since it prints the trajectory segment by segment between two consecutive centroids. The `Properties` command configures the line appearance. And we need the `ROI Manager` to keep every line segment on display. Is there any special attention we should pay to the `ROI Manager`?

```
makeLine(floor(objCenterX[i-1]),floor(objCenterY[i-1]),floor(objCenterX[i]),
    floor(objCenterY[i]));
run("Properties... ", "name=[] position=none stroke=Red width=2");
roiManager("Add");
```

:

An alternative is the `makeSelection` built-in function with `polyline` type, which does not need to plot line segment by segment but rakes the coordinates arrays.

Fig. 3.3 shows four examples of trajectories overlaid on the mask image. The source code can be found in the Session4 folder ("Tracking_measurements.ijm"). *How to plot multiple trajectories in one goal* is left for those who are interested to practice yourselves after the course. Also, The Fiji built-in functions "Overlay" is an alternative to display trajectories. We will also leave it for your own exploration.

3.4.2 Plot displacements or velocity

Examining the object displacements or velocity changes is another interesting measurement for tracking studies. Since we have already obtained the centroid position at each time frame, then the displacement of the centroid point from one time point to the next is the distance between the two positions. The displacement is the square root of the sum of squared difference between each coordinates of the points, which is calculated as shown

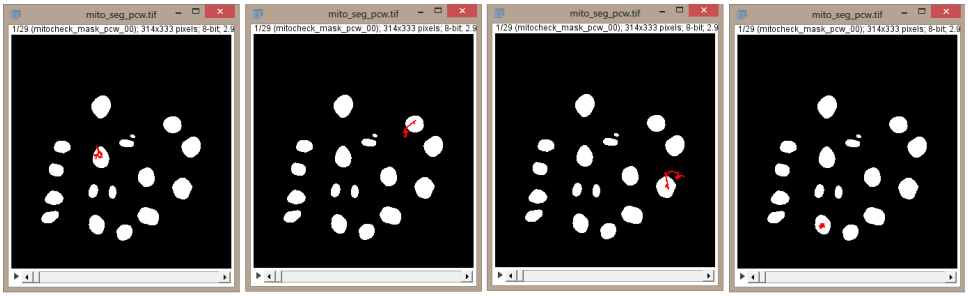


FIGURE 3.3: Example trajectories (marked in *red* lines) of object centroid movements.

in line 3 of the code below, in our case. The built-in function `sqrt` calculates the square root of a given number. The displacements over time can be stored in an array, "`disp`", which starts with one array element and keeps growing. This is because we do not expect to know in advance the temporal span of each object. Line 6-13 shows how the array element grows. In the code a flag "`arrayOK`" is used, what is its use?

```
1  disp = newArray(1);
   arrayOK = false;
3
   current_disp = sqrt( (objCenterX[i]-objCenterX[i-1])*(objCenterX[i]-
       objCenterX[i-1]) + (objCenterY[i]-objCenterY[i-1])*(objCenterY[i]-
       objCenterY[i-1]) );
5  //store the current displacement into the arry "disp"
   if (arrayOK)
7  {
     disp = Array.concat(disp,current_disp);
9  }else
   {
11   Array.fill(disp,current_disp);
     arrayOK = true;
13 }
```

:

And if the time interval is known, the displacement divided by the time interval gives the velocity. ImageJ offers the **Plot** functions to create a window showing a plot. Use it like this:

```
1  //Set the right time frame - xCoord - for the plot.
   //startFr and endFr mark the frames when the current object shows up and
       finishes
3  xCoord = newArray(endFr-startFr+1);
   for(i=startFr;i<=endFr;i++)
5  {
     xCoord[i-startFr]=i;
7  }
   Plot.create("Fancier Plot", "Frame", "Displacements (pixel)");
```

```
//Set the display range of each axis.
//maxY is the maximum displacement of the current object from array disp
Plot.setLimits(0, nSlices, 0, maxY+1);
Plot.setLineWidth(2);
Plot.setColor("lightGray");
Plot.add("line", xCoord, disp);
Plot.setColor("red");
Plot.add("circles", xCoord, disp);
Plot.show();
```

:

And Fig. 3.4 shows four examples of such plots from the corresponding objects shown in Fig. 3.3. Despite of the noisy profile, the differences in terms of amplitude and pattern are discernible.

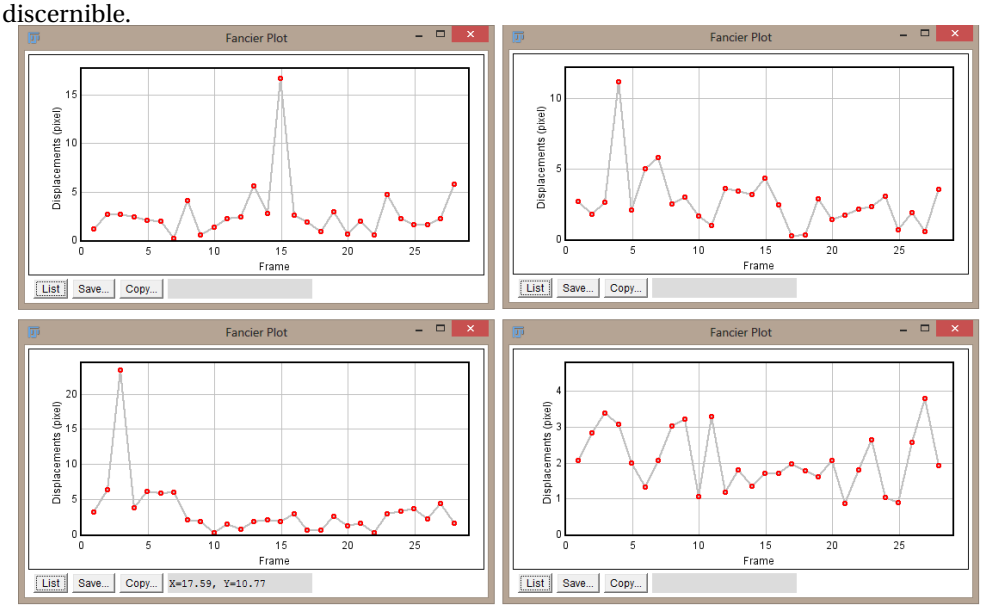


FIGURE 3.4: Example displacements of centroids from the four objects as in Fig. 3.3 (order: *left* to *right* and *top* to *bottom*).

3.5 KNIME solution

Open the workflow [Session4_Tracking]:

3.5.1 Image processing

By default the loaded image data provides the dimensions X, Y, Z. We can use the `Set Image Metadata || Label of dimension 2: Time` node in order to change

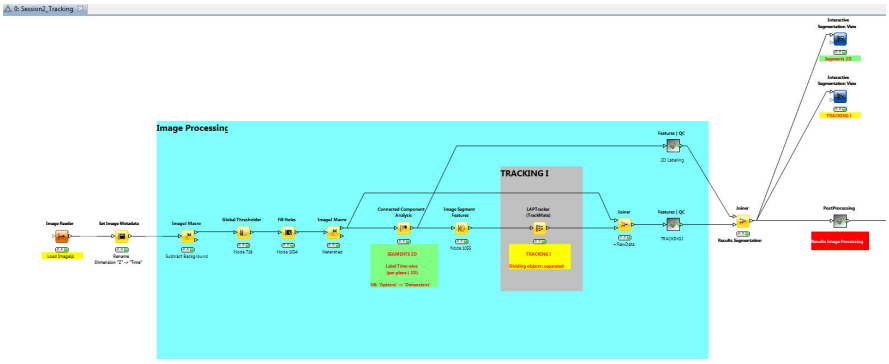


FIGURE 3.5: Image Processing part of the KNIME workflow “Session4_Tracking".

the metadata subset *Z* to *Time* (Z is our tracking, i.e. time dimension). After applying basic and already known image processing steps the `Connected Component Analysis` node is used in the same configuration compared to the sessions before, meaning that each object is identified separately plane by plane (Z / Time dimension). In order to obtain the object tracks we have to identify the same objects between different planes, i.e. we have to assign the same label for an object during Z / Time. This can be done using the `LAPTracker` node (Fig. 3.6):

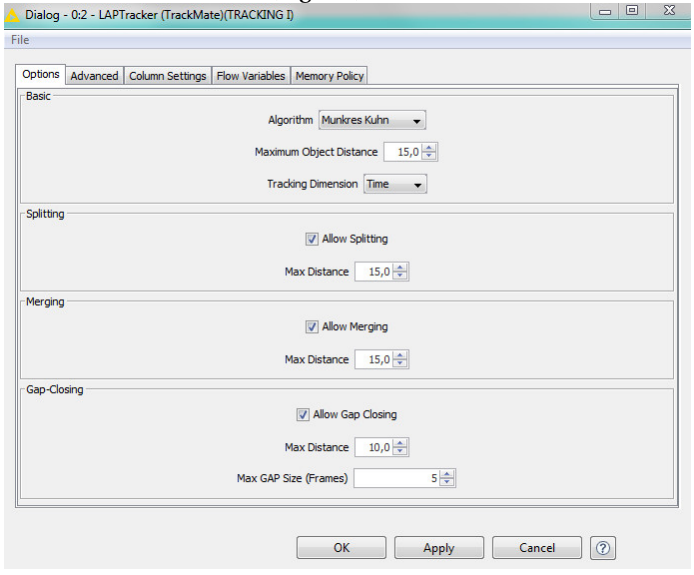


FIGURE 3.6: Configuration dialog of the `LAPTracker` node.

The configuration is very simple. The tracking dimension has to be set to *Time* and since we have to deal with dividing objects (nuclei) we have to enable the *Allow Splitting* and *Allow Merging* flags. The *Allow Gap Closing* flag considers objects which are not visible continuously during time (e.g. out of focus or outside the ROI for a certain number of Z planes / Time Points). You can set the maximum distance you would like an object to allow to displace during this time (*Max Distance*) and the number of maximum time points the object is not visible (*Max GAP Size (Frames)*).

Please note: Choose your settings very accurate! The best is to try different settings and to check which setting fits best to your data set / objects.

By comparing the outcome of the `Connected Component Analysis` and `LAPTracker` nodes you can directly see the difference in the object's / label assignment. The `Connected Component Analysis` node provides different labels (cp. colors) for the same nucleus between different time points. In contrast, the same nucleus has the same label in the outcome of the `LAPTracker` node, i.e. this object is tracked during Z / Time.

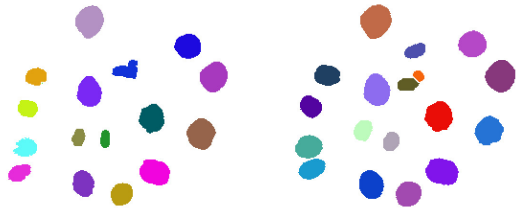


FIGURE 3.7: Result of the `Connected Component Analysis` node: The nuclei have different labels (cp. colors) between different time points (left: time point 1; right: time point 2), i.e. the objects are considered as being different.

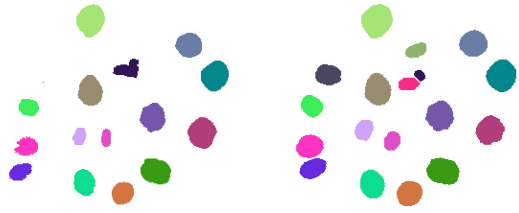


FIGURE 3.8: Result of the `LAPTracker` node: The nuclei have the same label (cp. colors) between different time points (left: time point 1; right: time point 2), i.e. the objects are considered as being identical and thus tracked during Z / Time.

With the outcome of the `LAPTracker` node we now obtained the tracking of the nuclei.

Please note: In case that an object divides into two parts, the `LAPTracker` node assigns two new labels to these segments, i.e. that we do not have access to information related to e.g. cell division time points. This can be solved by including the Add-On solution of the workflow (see below: 3) Add-On solution: a closer look).

In the following MetaNodes *Features | QC* we calculate the object features of the 2D (`Connected Component Analysis` node) and 2D+Time (`LAPTracker` node) segmentation.

Basically we are finished. With a (e.g.) `Segment Feature` node the tracks could be identified and written to a table.

Nevertheless, we are in addition interested in the object displacement and velocity, i.e. that we need the object information for each time point for the corresponding object track. This processing step is performed via the following *PostProcessing* MetaNode:

The entire information is already accessible in the result of the `Connected Component Analysis` and `LAPTracker` node respectively. We simply have to combine this information in a reasonable way:

The single time point information can be extracted easily by applying the `Image Segment Features` node to the 2D segmentation.

The information of the (different labeled) segments correspond to a certain track and can be extracted by combining the 2D and 2D+Time segmentation using the `Labeling Arithmetic` node. Again we use the concept of *LabelDependencies* within the `Image Segment Features` node. Resulting we obtain a list containing the information which 2D label (object) corresponds to which 2D+Time track (Fig. 3.9):

Row ID	Label [Single Time Point]	TRACK ID	X [Single Time Point]	Y [Single Time Point]	Time [...	Max [Si...	Mean [...
mitocheck_sm...	2D_171	TRACKING-I_Track: 11	120.855	115.701	10	82	44.604
mitocheck_sm...	2D_172	TRACKING-I_Track: 17	181.647	119.579	10	46	27.754
mitocheck_sm...	2D_170	TRACKING-I_Track: 23	142.758	80.226	10	81	44.285
mitocheck_sm...	2D_508	TRACKING-I_Track: 10	122.331	272.47	20	63	36.462
mitocheck_sm...	2D_509	TRACKING-I_Track: 9	174.877	286.542	20	66	37.386
mitocheck_sm...	2D_270	TRACKING-I_Track: 1	39.57	224.114	12	42	26.432
mitocheck_sm...	2D_271	TRACKING-I_Track: 13	176.416	157.292	8	26	19.885
mitocheck_sm...	2D_501	TRACKING-I_Track: 31	194.748	245.913	20	56	33.773
mitocheck_sm...	2D_71	TRACKING-I_Track: 5	196.907	195.385	5	81	52.071
mitocheck_sm...	2D_264	TRACKING-I_Track: 18	261.441	202.01	12	46	31.62
mitocheck_sm...	2D_640	TRACKING-I_Track: 3	115.441	206.433	14	40	25.607
mitocheck_sm...	2D_500	TRACKING-I_Track: 18	260.744	204.719	21	53	35.005
mitocheck_sm...	2D_263	TRACKING-I_Track: 7	71.506	133.931	8	39	27.416
mitocheck_sm...	2D_70	TRACKING-I_Track: 8	63.562	182.81	5	36	24.538
mitocheck_sm...	2D_503	TRACKING-I_Track: 19	254.374	247.642	20	54	33.969
mitocheck_sm...	2D_262	TRACKING-I_Track: 5	212.159	193.785	12	77	44.307
mitocheck_sm...	2D_642	TRACKING-I_Track: 1	38.483	222.841	14	43	26.656
mitocheck_sm...	2D_502	TRACKING-I_Track: 3	116.47	206.901	21	57	32.125
mitocheck_sm...	2D_261	TRACKING-I_Track: 17	187.981	123.038	8	47	32.1
mitocheck_sm...	2D_641	TRACKING-I_Track: 24	193.401	213.938	14	50	32.033
mitocheck_sm...	2D_505	TRACKING-I_Track: 0	66.321	251.149	20	45	27.746
mitocheck_sm...	2D_268	TRACKING-I_Track: 12	126.364	166.558	8	83	53.385
mitocheck_sm...	2D_504	TRACKING-I_Track: 1	45.296	219.176	21	52	32.74
mitocheck_sm...	2D_267	TRACKING-I_Track: 15	272.648	142.398	8	57	37.466
mitocheck_sm...	2D_507	TRACKING-I_Track: 2	224.993	265.601	20	73	37.619
mitocheck_sm...	2D_266	TRACKING-I_Track: 24	191.27	207.078	12	57	36.755
mitocheck_sm...	2D_506	TRACKING-I_Track: 4	153.471	232.333	21	43	28.581
mitocheck_sm...	2D_265	TRACKING-I_Track: 14	216.993	142.438	8	34	25

FIGURE 3.9: Outcome of the `Column Rename` node in the PostProcessing MetaNode: The column *Label [Single TimePoint]* provides the information for every time point. The column *TRACK ID* lists the corresponding object track.

After that we combine the two results using the `Joiner || Joiner Setting: Label , TRACK ID` node and we end up with a table containing the full results of the

image processing part (Fig. 3.10):

Row ID	Label	Bitmask	Label [Single Time Point]	X [Singl...	Y [Singl...	Time [Si...	X [Cen...	Y [Cen...	Time [...	X [Dim...	Y [Dim...	Max [Si...	Mean [...	Max Int.
mitocheck_sm...	TRACKING-1_Track: 0		2D_19	54.199	259.419	0	64.341	251.199	29	40	34	31	22.568	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_89	60.162	256.341	1	64.341	251.199	29	40	34	29	19.887	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_42	62.504	254.736	2	64.341	251.199	29	40	34	30	20.925	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_56	67.718	255.284	3	64.341	251.199	29	40	34	29	22.08	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_244	69.111	255.234	4	64.341	251.199	29	40	34	30	22.358	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_96	69.632	252.411	5	64.341	251.199	29	40	34	31	23.101	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_115	69.885	251.027	6	64.341	251.199	29	40	34	32	22.813	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_137	69.795	252.093	7	64.341	251.199	29	40	34	31	23.475	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_292	67.056	251.544	8	64.341	251.199	29	40	34	32	23.488	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_164	69.158	251.075	9	64.341	251.199	29	40	34	33	23.505	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_191	68.256	250.213	10	64.341	251.199	29	40	34	33	21.277	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_224	65.727	249.979	11	64.341	251.199	29	40	34	37	23.273	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_277	66.228	249.645	12	64.341	251.199	29	40	34	36	25.442	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_330	66.323	250.169	13	64.341	251.199	29	40	34	37	26.016	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_645	63.61	249.774	14	64.341	251.199	29	40	34	41	26.181	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_341	66.121	247.679	15	64.341	251.199	29	40	34	40	27.681	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_383	65.769	248.874	16	64.341	251.199	29	40	34	41	25.621	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_396	63.439	248.414	17	64.341	251.199	29	40	34	43	28.942	57
mitocheck_sm...	TRACKING-1_Track: 0		2D_440	64.017	249.991	18	64.341	251.199	29	40	34	43	26.308	57

FIGURE 3.10: Resulting table of the image processing part: Object tracks (column: *Label*) with corresponding information for each time point (column: *Label [Single TimePoint]*).

3.5.2 Statistics: Calculating object displacement and velocity

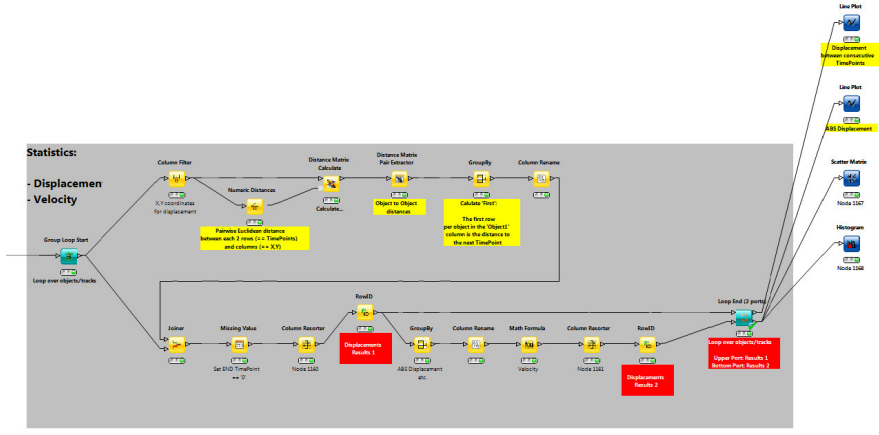


FIGURE 3.11: Statistics part of the KNIME workflow *Session4_Tracking*.

As a result of the image processing part we have the information of the object tracks and the corresponding information for each time point of each of the object tracks. In order to identify object movement we calculate the X, Y displacements between consecutive time points individual for each object track. This can be done by using loops: The `Group Loop Start || Include: *Label*` node picks out the rows contain-

ing the same label, i.e. object track, and loops over each object (Check the output of the `Group Loop Start` node).

In order to calculate the displacement we can (e.g.) use so called *Distance Matrix* nodes. This is a group of three nodes: The pair `Numeric Distances || Include: X [Single TimePoint] | Include Y [Single TimePoint]` and `Distance Matrix Calculate` provides the distance in Euclidean space between each row, i.e. in our case between each time point. With the `Distance Matrix Pair Extractor` node we obtain a list containing two columns for the objects (i.e. single time point label) and the distance (X, Y) between each of these time points.

Row ID	S Object1	S Object2	D Distance
Row0	2D_19	2D_89	6.711
Row1	2D_19	2D_42	9.534
Row2	2D_19	2D_56	14.138
Row3	2D_19	2D_244	15.488
Row4	2D_19	2D_96	16.95
Row5	2D_19	2D_115	17.79
Row6	2D_19	2D_137	17.232
Row7	2D_19	2D_292	15.077
Row8	2D_19	2D_164	17.129
Row9	2D_19	2D_191	16.804
Row10	2D_19	2D_224	14.9
Row11	2D_19	2D_277	15.499
Row12	2D_19	2D_330	15.251
Row13	2D_19	2D_645	13.476
Row14	2D_19	2D_341	16.732
Row15	2D_19	2D_383	15.655
Row16	2D_19	2D_396	14.37

FIGURE 3.12: Outcome of the `Distance Matrix Pair Extractor` node.

Since we are only interested in the distance between two consecutive object movements, we only need the first value of each of the objects in the *Object1* column. This can simply be done using a `GroupBy || Options: *Distance*: First` node (Fig. 3.13):

After that we perform some post-processing and obtain the final results via the two output ports of the `Loop End` node: The upper port lists the information per single time point for each of the object tracks; the lower port lists the results per track.

3.5.3 Add-On solution: A closer look

This part will not be explained in detail! (For the interested user)

The main idea is the following: In this data set we have objects which divide. Now it could be of great interest to assign *root* and *daughter* cells, meaning to identify which

Row ID	Object1	X,Y [Displacement between TimePoints]
Row3	2D_19	6.711
Row26	2D_89	2.839
Row13	2D_42	5.244
Row18	2D_56	1.393
Row6	2D_244	2.871
Row27	2D_96	1.406
Row0	2D_115	1.07
Row1	2D_137	2.794
Row8	2D_292	2.154
Row2	2D_164	1.248
Row4	2D_191	2.54
Row5	2D_224	0.602
Row7	2D_277	0.532
Row9	2D_330	2.742
Row21	2D_645	3.27
Row10	2D_341	1.245
Row11	2D_383	2.375
Row12	2D_396	1.679
Row14	2D_440	2.747
Row15	2D_450	1.819
Row16	2D_505	2.499
Row17	2D_523	1.653
Row19	2D_579	5.598
Row20	2D_589	2.348
Row22	2D_666	1.207
Row24	2D_754	1.323
Row25	2D_804	2.495
Row23	2D_710	1.852

FIGURE 3.13: Result table of the displacement calculation: The column X, Y [...] lists the displacement in X, Y between the *Object* with label x, i.e. to the next consecutive time point.

FIGURE 3.14: Add-On part of the KNIME workflow *Session4_Tracking*.

object is a result of a cell division process.

The most important step is done by using the `Connected Component Analysis || Dimensions: X, Y, Time` node. In this configuration this node acts like the `LAPTracker` node; with one important difference: The labels of dividing objects stay the same! By combining this information with the information of the 2D (`Connected Component Analysis` node) and 2D+Time (`LAPTracker` node) of the upper branch, we finally obtain our *root* and *daughter* cells; cp. outcome of *Final Results*

table:

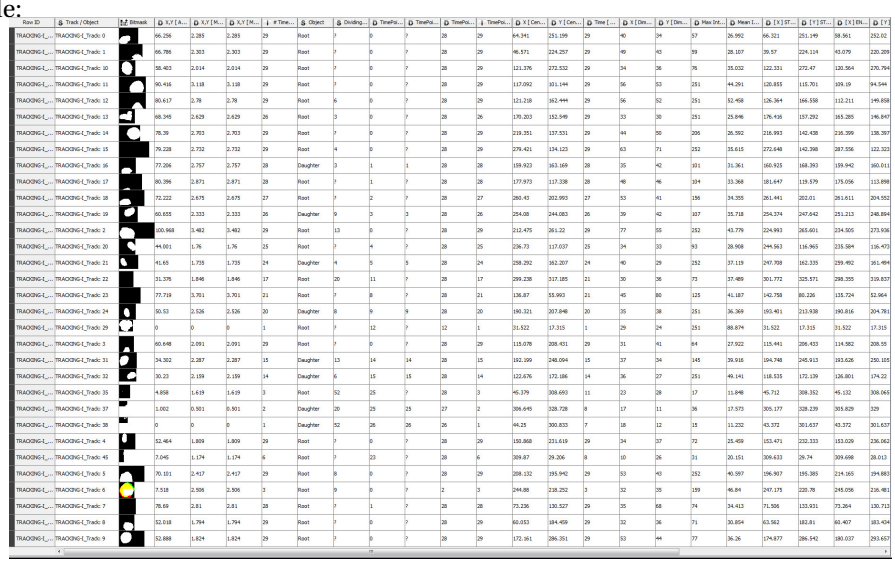


FIGURE 3.15: Final results table.

3.6 Tips and tools

- ImageJ Built-in Macro Functions: http://rsbweb.nih.gov/ij/developer/macro/functions.html
- Built-in functions used: getDimensions, getStatistics, newArray, Array.fill, Array.concatenate, setSlice, nSlices, nResults, getResult, makeLine, Plot
- EuBIAS 2013 - BioImage Data Analysis Course: http://eubias2013.irbbarcelona.org/bias-2-course

3.7 Groups close by which work on similar problems

- Dr. Karl Rohr's group, Biomedical Computer Vision Group (http://www.bioquant.uni-heidelberg.de/?id=322), located at Bioquant
- Prof. Fred Hamprecht's group, Multidimensional Image Processing Group (http://hci.iwr.uni-heidelberg.de/MIP/), located at HCI, Speyererstr. 6
- Dr. Christian Conrad's group, Intelligent Imaging Group (http://ibios.dkfz.de/tbi/index.php/intelligent-imaging/people/94-cconrad), located at Bioquant

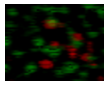
3.8 References

[1] B Neumann, T Walter, J-K Hériché, J Bulkescher, H Erfle, C Conrad, P Rogers, I Poser, M Held, U Liebel, C Cetin, F Sieckmann, G Pau, R Kabbe, A Wünsche, V Satagopam, M H A Schmitz, C Chapuis, D W Gerlich, R Schneider, R Eils, W Huber, J-M Peters, A A Hyman, R Durbin, R Pepperkok, and J Ellenberg. Phenotypic profiling of the human genome by time-lapse microscopy reveals cell division genes. *Nature*, 464:721–727, 2010.

4

3D Object based colocalization

We would like to thank Dr. Ke Peng (Virology department, University of Heidelberg) for sharing some of his datasets and accepting to expose part of the methodology involved in his own research work.



4.1 Aim

In this project we will first segment objects of interest; then implement an ImageJ macro to analyze 3D object based colocalization; finally how the colocalization results can be visualized will be discussed.

4.2 Introduction

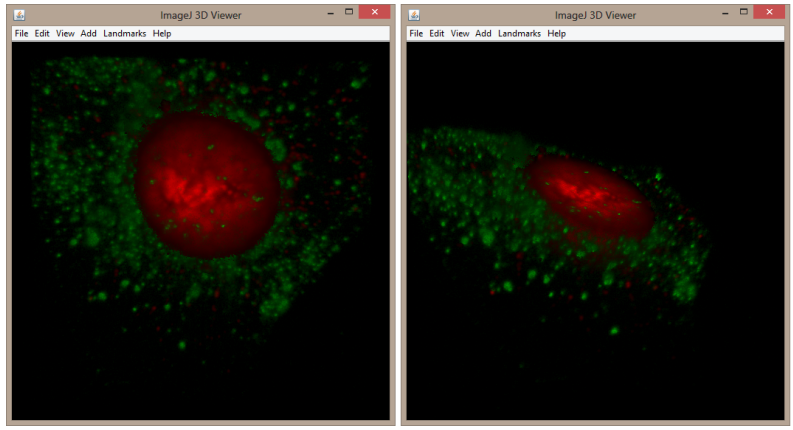


FIGURE 4.1: The Hela cells dataset (with two channels) from two views.

Subcellular structures interact in numerous ways, which depend on spatial proximity or spatial correlations between the interacting structures. Colocalization analysis aims at finding such correlations, providing hints of potential interactions. If the structures only have simple spatial overlap with one another, it is called *cooccurrence*; If they not only overlap but also codistribute in proportion, it is then *correlation*. Colocalization may be evaluated visually, quantitatively, and statistically:

1. It may be identified by superimposing two images and inspecting the appearance of the combined color. For example, colocalization of red and green structures can appear yellow. However, this intuitive method can work only when the intensity levels of the two images are similar (see a detailed example in [2]). Scatter plot of pixel intensities from the two images also qualitatively indicates colocalization, e.g. the points form a straight line if the two structures correlate. But visual evaluation does not tell the degree of colocalization, nor if it is true colocalization at all.

2. In general, two categories of quantitative approaches to colocalization analysis can be found: intensity based correlation methods and object based methods. Intensity based methods compute global measures about colocalization, using the correlation information of intensities of two channels. Several review papers have been published during the last decade, where coefficients' meaning, interpretation, guide

of use, and examples for colocalization are given [1, 2, 8]. Tools for quantifying these measures can be found in many image analysis open-source and commercial software packages, to name just a few: Fiji's JACoP plugin and Coloc 2, CellProfiler, BioImageXD, Huygens, Imaris, Metamorph, Volocity. Most object-based colocalization methods first segment and identify objects, and then account for objects' inter-distances to analyze possible colocalization. Usually, two objects are considered colocalized, if the centroids of the objects are within certain distance [1, 5], or if two objects with certain percentage of area/volume overlap [6, 7].

3. Colocalization studies generally should perform some statistical analysis, in order to interpret whether the found cooccurrence or correlation is just a random coincidence or a true colocalization. A common method is Monte-Carlo simulations [3] but it is computationally expensive. Recently a new analytical statistics method based on Ripley's K function is proposed and included as an Icy plugin, StatColoc [4].

Dataset: Virologists often need to answer the questions of when and where the viral replication happens and the relevant virus-host interactions. The dataset (see Fig. 4.1 *top* as an example) we are using in this session is Hela cells imaged with a Spinning disk microscope. Z serial images were acquired in three channels, from which two are used: channel 1 (C1, red) shows viral DNA in high contrast and channel 3 (C3, green) shows viral particles in high contrast (a viral structural protein). The high contrast signals either come from synthetic dyes or fluorescent protein. The goal is to identify the viral particles that have synthesized viral DNA indicating such structures represent replicating viral particles and potentially the viral replication sites. Thus, the identification can be achieved through a colocalization analysis between the objects in these two channels.

4.3 KNIME solution

Open the workflow [Session5_Colocalisation_NonPointLike] (Fig. 4.2):

4.3.1 Image processing

The *Image PreProcessing* part combines the raw data to 3D data stacks for further processing. Within *Image Processing* we use already known, straight forward methods to segment the spots. Important to mention is, that due to the quality of the image raw data we here use the `Local Thresholding` node with subsequent morphological operations in order to remove noise. This procedure turned out to fit best for this data set. Alternatives would be the well-known `Global Thresholder` or `Spot Detection` nodes.

The result of the image processing part looks as follows (Fig. 4.3):

4.3.2 Identify colocalizations and corresponding spot areas

In order to identify colocalizations it would be enough to use the already known `Labeling Arithmetic || Labeling Operation Method: AND` node which returns only

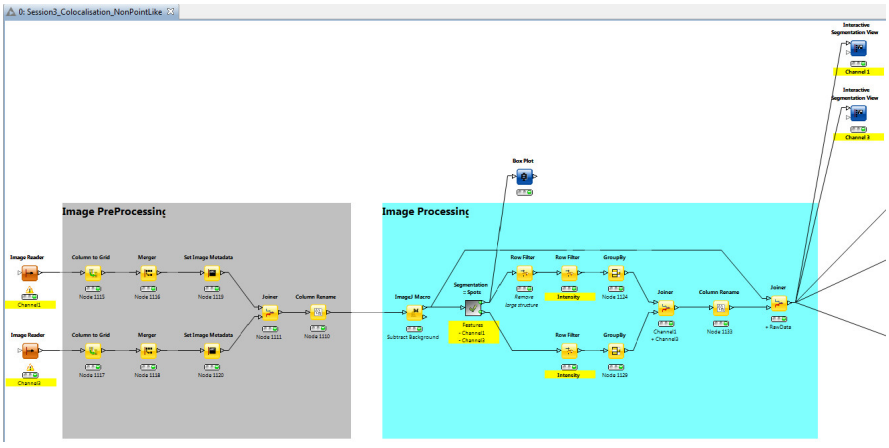


FIGURE 4.2: Image Processing part of the KNIME workflow “Session5_Colocalisation_NonPointLike".

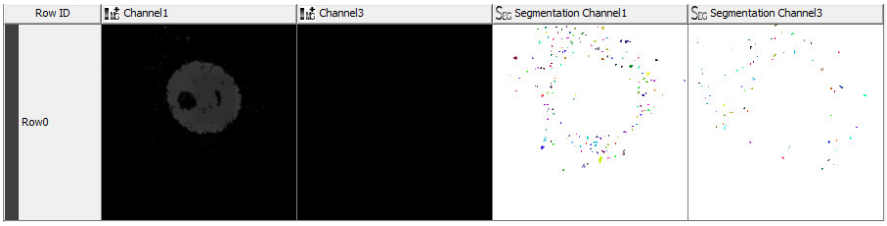


FIGURE 4.3: Result table of the image processing part including the image raw data and segmented spots in 3D.

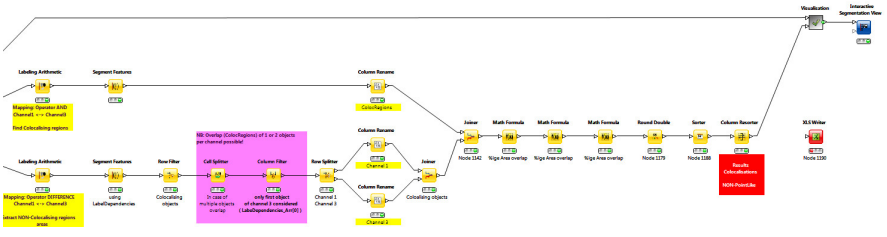
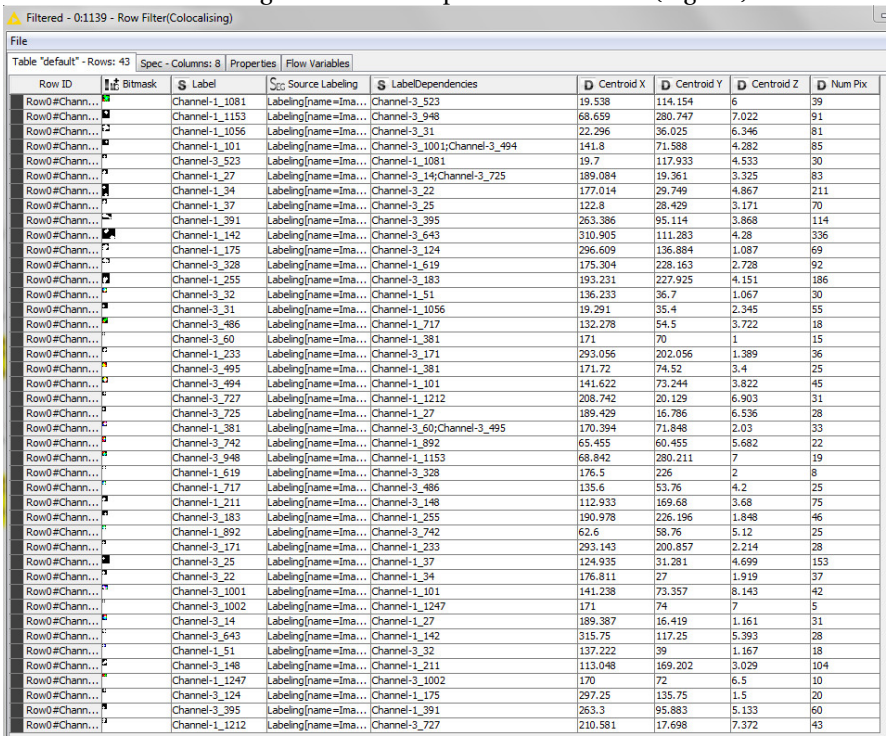


FIGURE 4.4: Statistics part of the KNIME workflow “Session5_Colocalisation_NonPointLike".

the segments which are overlapping between the two data stacks (upper branch).

However, we are in addition interested in the overlapping regions in terms of areas. I.e., that we have to identify the corresponding spot regions of the two channels which colocalize. For this we use again the `Labeling Arithmetic || Labeling Operation Method: DIFFERENCE` node applying the configuration *DIFFERENCE*. This operation returns the regions which do NOT overlap and thus provides the information we are interested in. By using the *LabelDependencies* feature we can now identify the non-overlapping regions of the colocalizing spots.

It turns out that some regions have multiple colocalizations (Fig. 4.5):



Filtered - 0:1139 - Row Filter(Colocalising)

File

Table "default" - Rows: 43 | Spec - Columns: 8 | Properties | Flow Variables

Row ID	Bitmask	Label	Source Labeling	LabelDependencies	Centroid X	Centroid Y	Centroid Z	Num Pix
Row0#Chann...		Channel-1_1081	Labeling[name=Ima...	Channel-3_523	19.538	114.154	6	39
Row0#Chann...		Channel-1_1153	Labeling[name=Ima...	Channel-3_948	68.659	280.747	7.022	91
Row0#Chann...		Channel-1_1056	Labeling[name=Ima...	Channel-3_31	22.296	36.025	6.346	81
Row0#Chann...		Channel-1_101	Labeling[name=Ima...	Channel-3_1001;Channel-3_494	141.8	71.588	4.282	85
Row0#Chann...		Channel-3_523	Labeling[name=Ima...	Channel-1_1081	19.7	117.933	4.533	30
Row0#Chann...		Channel-1_27	Labeling[name=Ima...	Channel-3_14;Channel-3_725	189.084	19.361	3.325	83
Row0#Chann...		Channel-1_34	Labeling[name=Ima...	Channel-3_22	177.014	29.749	4.867	211
Row0#Chann...		Channel-1_37	Labeling[name=Ima...	Channel-3_25	122.8	28.429	3.171	70
Row0#Chann...		Channel-1_391	Labeling[name=Ima...	Channel-3_395	263.386	95.114	3.868	114
Row0#Chann...		Channel-1_142	Labeling[name=Ima...	Channel-3_643	310.905	111.283	4.28	336
Row0#Chann...		Channel-1_175	Labeling[name=Ima...	Channel-3_124	296.609	136.884	1.087	69
Row0#Chann...		Channel-3_328	Labeling[name=Ima...	Channel-1_619	175.304	228.163	2.728	92
Row0#Chann...		Channel-1_255	Labeling[name=Ima...	Channel-3_183	193.231	227.925	4.151	186
Row0#Chann...		Channel-3_32	Labeling[name=Ima...	Channel-1_51	136.233	36.7	1.067	30
Row0#Chann...		Channel-3_31	Labeling[name=Ima...	Channel-1_1056	19.291	35.4	2.345	55
Row0#Chann...		Channel-3_486	Labeling[name=Ima...	Channel-1_717	132.278	54.5	3.722	18
Row0#Chann...		Channel-3_60	Labeling[name=Ima...	Channel-1_381	171	70	1	15
Row0#Chann...		Channel-1_233	Labeling[name=Ima...	Channel-3_171	293.056	202.056	1.389	36
Row0#Chann...		Channel-3_495	Labeling[name=Ima...	Channel-1_381	171.72	74.52	3.4	25
Row0#Chann...		Channel-3_494	Labeling[name=Ima...	Channel-1_101	141.622	73.244	3.822	45
Row0#Chann...		Channel-3_727	Labeling[name=Ima...	Channel-1_1212	208.742	20.129	6.903	31
Row0#Chann...		Channel-3_725	Labeling[name=Ima...	Channel-1_27	189.429	16.786	6.536	28
Row0#Chann...		Channel-1_381	Labeling[name=Ima...	Channel-3_60;Channel-3_495	170.394	71.848	2.03	33
Row0#Chann...		Channel-3_742	Labeling[name=Ima...	Channel-1_892	65.455	60.455	5.682	22
Row0#Chann...		Channel-3_948	Labeling[name=Ima...	Channel-1_1153	68.842	280.211	7	19
Row0#Chann...		Channel-1_619	Labeling[name=Ima...	Channel-3_328	176.5	226	2	8
Row0#Chann...		Channel-1_717	Labeling[name=Ima...	Channel-3_486	135.6	53.76	4.2	25
Row0#Chann...		Channel-1_211	Labeling[name=Ima...	Channel-3_148	112.933	169.68	3.68	75
Row0#Chann...		Channel-3_183	Labeling[name=Ima...	Channel-1_255	190.978	226.196	1.848	46
Row0#Chann...		Channel-1_892	Labeling[name=Ima...	Channel-3_742	62.6	58.76	5.12	25
Row0#Chann...		Channel-3_171	Labeling[name=Ima...	Channel-1_233	293.143	200.857	2.214	28
Row0#Chann...		Channel-3_25	Labeling[name=Ima...	Channel-1_37	124.935	31.281	4.699	153
Row0#Chann...		Channel-3_22	Labeling[name=Ima...	Channel-1_34	176.811	27	1.919	37
Row0#Chann...		Channel-3_1001	Labeling[name=Ima...	Channel-1_101	141.238	73.357	8.143	42
Row0#Chann...		Channel-3_1002	Labeling[name=Ima...	Channel-1_1247	171	74	7	5
Row0#Chann...		Channel-3_14	Labeling[name=Ima...	Channel-1_27	189.387	16.419	1.161	31
Row0#Chann...		Channel-3_643	Labeling[name=Ima...	Channel-1_142	315.75	117.25	5.393	28
Row0#Chann...		Channel-1_51	Labeling[name=Ima...	Channel-3_32	137.222	39	1.167	18
Row0#Chann...		Channel-3_148	Labeling[name=Ima...	Channel-1_211	113.048	169.202	3.029	104
Row0#Chann...		Channel-1_1247	Labeling[name=Ima...	Channel-3_1002	170	72	6.5	10
Row0#Chann...		Channel-3_124	Labeling[name=Ima...	Channel-1_175	297.25	135.75	1.5	20
Row0#Chann...		Channel-3_395	Labeling[name=Ima...	Channel-1_391	263.3	95.883	5.133	60
Row0#Chann...		Channel-1_1212	Labeling[name=Ima...	Channel-3_727	210.581	17.698	7.372	43

FIGURE 4.5: Multiple colocalizations: See column “LabelDependencies".

To simplify the story, we only focus on one colocalization partner. By using the `Cell Splitter` node we split the column *LabelDependencies* into two new columns with separated label entries *LabelDependencies_Arr[0]* and *LabelDependencies_Arr[1]* and consider only *LabelDependencies_Arr[0]* in the following.

The two branches are then combined by the `Joiner` node and resulting we obtain a table including the colocalization regions and the corresponding spot areas of the two channels (Fig. 4.6):

By performing some basic post-processing we obtain the final results including the spot labels, the colocalization region and the corresponding areas.

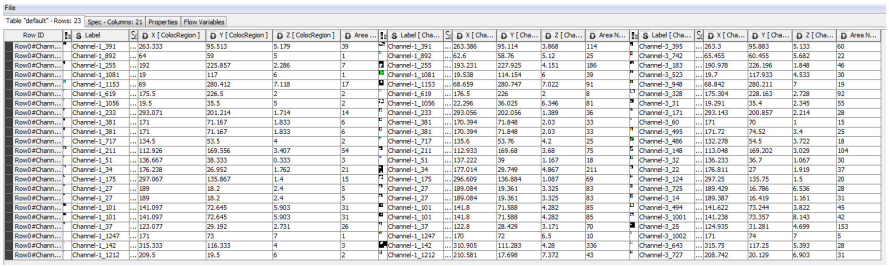


FIGURE 4.6: Result of the `Joiner` node after combining the two branches.

4.3.3 Add-On: Point-like colocalization

Very often we have to deal with point-like objects or the information needed can be defined by the distance of the center of mass of the objects. In this case a very simple solution can be applied:

Open the workflow [Session5_Colocalisation_NonPointLike_PointLike] (Fig. 4.7):

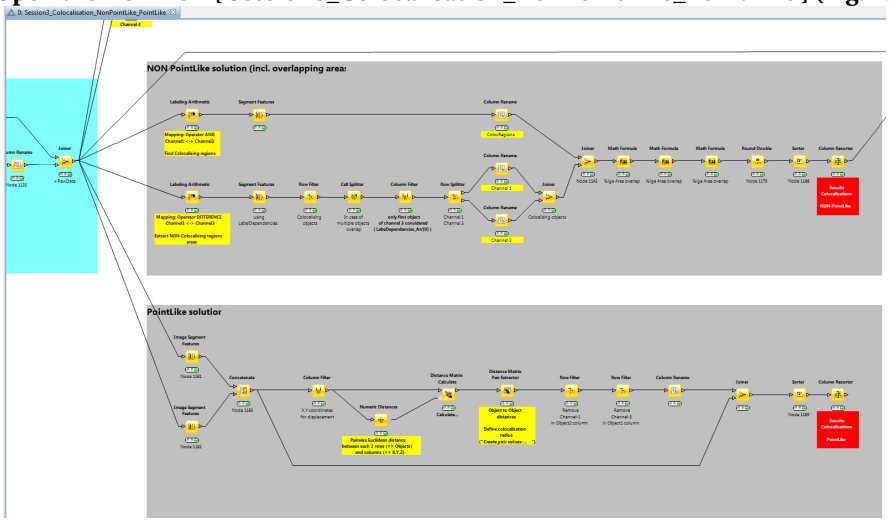


FIGURE 4.7: Statistics part of the KNIME workflow *Session5_Colocalisation_NonPointLike_PointLike*.

We simply calculate the X, Y, Z coordinates of the spots via the `Image Segment Features` nodes and use the same method as applied in the tracking session by using *Distance Matrix*. We can define a colocalization radius of e.g. 5 by configuring the `Distance Matrix Pair Extractor || Create pair for values: <= 5` node (Fig. 4.8):

The final result table provides the following colocalizing objects (cp. result table: non-

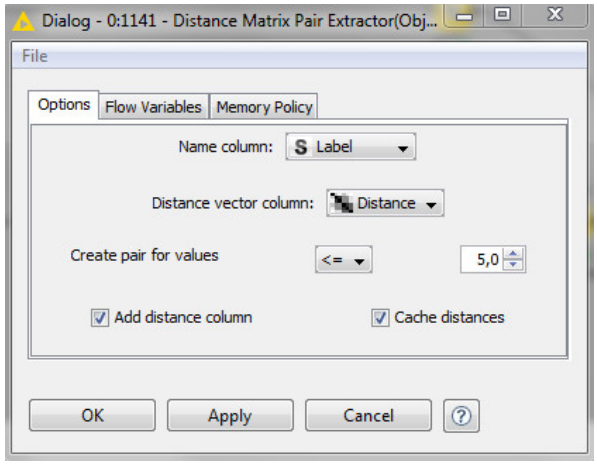


FIGURE 4.8: Configuration dialog of the `Distance Matrix Pair Extractor` node: Define a colocalization radius of (e.g.) 5 by simply creating pairs for values <= 5.

point-like solution!) (Fig. 4.9):

Table "default" - Rows: 38 | Spec - Columns: 8 | Properties | Flow Variables

Row ID	Object1	Object2	Distance between Objects	Max	Mean	Num Pix	Bitmask	Source Labeling
Row22_Row0...	Channel-1_101	Channel-3_494	1.728	1,910	1,220.847	85		Labeling[name=Image[...
Row23_Row0...	Channel-1_101	Channel-3_1001	4.283	1,910	1,220.847	85		Labeling[name=Image[...
Row10_Row0...	Channel-1_1081	Channel-3_523	4.057	1,974	785.359	39		Labeling[name=Image[...
Row17_Row0...	Channel-1_1153	Channel-3_948	0.567	5,951	1,286.901	91		Labeling[name=Image[...
Row89_Row0...	Channel-1_1212	Channel-3_727	3.085	2,007	1,035.884	43		Labeling[name=Image[...
Row87_Row0...	Channel-1_1247	Channel-3_495	4.35	1,030	778.2	10		Labeling[name=Image[...
Row88_Row0...	Channel-1_1247	Channel-3_1002	2.291	1,030	778.2	10		Labeling[name=Image[...
Row34_Row0...	Channel-1_143	Channel-3_520	3.453	2,239	1,523.75	28		Labeling[name=Image[...
Row36_Row0...	Channel-1_175	Channel-3_124	1.367	8,640	2,031.913	69		Labeling[name=Image[...
Row84_Row0...	Channel-1_202	Channel-3_137	4.795	1,256	674.754	57		Labeling[name=Image[...
Row74_Row0...	Channel-1_211	Channel-3_148	0.816	4,366	1,673.6	75		Labeling[name=Image[...
Row76_Row0...	Channel-1_212	Channel-3_152	3.963	1,692	1,167.471	17		Labeling[name=Image[...
Row82_Row0...	Channel-1_227	Channel-3_157	4.22	2,153	1,236.702	47		Labeling[name=Image[...
Row60_Row0...	Channel-1_233	Channel-3_171	1.458	1,234	743.222	36		Labeling[name=Image[...
Row49_Row0...	Channel-1_251	Channel-3_328	3.046	1,035	727.6	5		Labeling[name=Image[...
Row47_Row0...	Channel-1_255	Channel-3_183	3.656	3,319	1,663.742	186		Labeling[name=Image[...
Row53_Row0...	Channel-1_269	Channel-3_191	4.555	1,339	767.982	55		Labeling[name=Image[...
Row25_Row0...	Channel-1_27	Channel-3_725	4.13	1,445	727.024	83		Labeling[name=Image[...
Row26_Row0...	Channel-1_27	Channel-3_14	3.665	1,445	727.024	83		Labeling[name=Image[...
Row28_Row0...	Channel-1_34	Channel-3_22	4.036	15,132	5,119.498	211		Labeling[name=Image[...
Row29_Row0...	Channel-1_37	Channel-3_25	3.877	1,945	926.514	70		Labeling[name=Image[...
Row64_Row0...	Channel-1_381	Channel-3_60	2.201	2,668	1,802.636	33		Labeling[name=Image[...
Row65_Row0...	Channel-1_381	Channel-3_495	3.282	2,668	1,802.636	33		Labeling[name=Image[...
Row31_Row0...	Channel-1_391	Channel-3_395	1.483	3,198	1,873.254	114		Labeling[name=Image[...
Row86_Row0...	Channel-1_51	Channel-3_32	2.506	1,070	825.389	18		Labeling[name=Image[...
Row56_Row0...	Channel-1_561	Channel-3_85	4.724	4,733	1,975.735	83		Labeling[name=Image[...
Row57_Row0...	Channel-1_562	Channel-3_875	4.78	1,451	807.08	50		Labeling[name=Image[...
Row61_Row0...	Channel-1_579	Channel-3_647	4.927	4,329	2,022.129	31		Labeling[name=Image[...
Row68_Row0...	Channel-1_580	Channel-3_1127	4.724	2,205	1,105.58	69		Labeling[name=Image[...
Row69_Row0...	Channel-1_580	Channel-3_774	2.855	2,205	1,105.58	69		Labeling[name=Image[...
Row79_Row0...	Channel-1_591	Channel-3_782	3.742	1,489	1,206.4	5		Labeling[name=Image[...
Row70_Row0...	Channel-1_619	Channel-3_328	2.577	1,171	941.25	8		Labeling[name=Image[...
Row83_Row0...	Channel-1_71	Channel-3_486	4.638	1,377	1,065.4	5		Labeling[name=Image[...
Row72_Row0...	Channel-1_717	Channel-3_992	4.305	1,425	758.32	25		Labeling[name=Image[...
Row73_Row0...	Channel-1_717	Channel-3_486	3.437	1,425	758.32	25		Labeling[name=Image[...
Row75_Row0...	Channel-1_718	Channel-3_1213	3.778	3,937	1,142.682	85		Labeling[name=Image[...
Row80_Row0...	Channel-1_892	Channel-3_742	3.367	1,239	700.24	25		Labeling[name=Image[...
Row44_Row0...	Channel-1_935	Channel-3_782	3.867	2,504	1,785.587	63		Labeling[name=Image[...

FIGURE 4.9: Final result table of the point-like colocalization.

4.4 Writing our own ImageJ macro for fully automatic colocalization

Many of you may have already been using tools e.g. the ImageJ plugin JACoP [1], Coloc2[1] for either object- or intensity-based colocalization analysis. Here, we will not use any of these, but try to write our own macro for fully automatic colocalization so that we could establish our own protocols and control settings, and at the same time practice more with macro scripting.

We have practiced in the previous sessions with: the function [`Plugins -> Macros -> Record`] to track the sequential operations that have been applied; cleaning up and converting the recorded macro commands to a functional macro file; and even a few ImageJ built-in functions. In this session, we will exploit a little bit more ImageJ Macro scripting, e.g. use more and different functions, construct a parameter setting interface window, etc.

In order to help better understanding the steps during the object-based colocalization, we will first use a synthetic dataset to test and build up the macro. And once our macro program is working, we will test it on the segmented dataset from the previous step. Fig. 4.10 shows two 3D views of the synthetic dataset with two channels, where channel 1 (*red*) has six objects and channel 2 (*blue*) seven. Each object in channel 1 has different level of spatial overlap with one of the objects in channel 2. The synthetic dataset can be found in the Session5 folder (C1_syn and C2_syn).

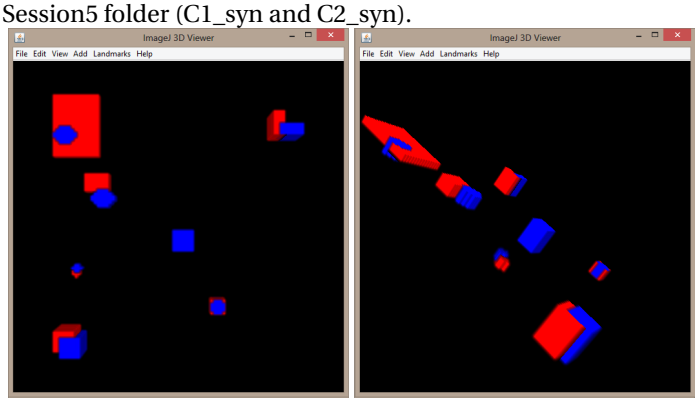


FIGURE 4.10: Synthetic 3D dataset from two views.

4.4.1 Filtering objects by size

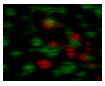
Often the segmentation contains objects that are not interesting for us such as noise or other structures. Since object-based methods concern individual objects, then we should apply some filtering criteria in order to discard them for further analysis. Such criteria could be, for example:

- (3D/2D/1D) size range of the object-of-interest (in each channel)

[1] http://fiji.sc/Coloc_2

- object shape, e.g. circularity[2], compactness[3]
- object location

It should be mentioned that this step greatly influences the colocalization measurements. We will discuss only size related filtering here. Our strategy consists of two steps: filtering in 3D, and then in 2D.



4.4.1.1 Filtering by 3D sizes

As we have already learned from the previous session, `3D Objects Counter` is able to do this. We can open the macro file "`S4_filtering.ijm`" and add steps in. After running it, we will be asked to specify the directories where the images from two channels are, and also a directory to store results. After that, we will see a window with many parameters to setup. Let's skip these first, and comment on them at later steps. So now let's select the image of channel 1, and then run [`Analyze -> 3D Object Counter`], in the popup window there are two parameters of our interest, `Min` and `Max` in the `Size filter` field. Let's suppose that the object of interest should have a size of minimum 3 voxels and maximum 10 voxels in each of the three dimensions, resulting in object volume of size `Min`=27 and `Max`=1000. This filtering step removes the smallest object from both channels. Although they may seem to overlap with objects in the other channel, they are likely to be e.g. noise and their spatial co-occurrence could be coming from randomly distributed particles/noises that are close to each other by chance. Since in this session we may have to produce many intermediate images, so it might be a good practice to `rename` these intermediate images. And if they will not be used any further, they might as well just be closed by running [`File -> Close`].

4.4.1.2 Filtering by 2D sizes

You may have noticed that this filtering criteria is not sufficient to remove the large object in channel 1 (Fig. 4.11 *left*). This is because e.g. the object is very thin in one or two axes and large in other(s) thus the total 3D size may be within the size range of the object of interest. In this case, it makes sense to have another filtering but in 2D rather than in 3D.

Option #1: For example, one possibility is to set a size range in the XY plane, and if needed also a maximum spanning size in the Z axis. In order to do this, we will use [`Analyze -> Analyze Particles`] and its `Size` parameter setting. So similarly we could set the value to be e.g. 9-100. Also, distinct size range can be employed for each channel if needed. Additionally, for some applications, the parameter `Circularity` could also be used. Since we would like to obtain the filtered image, thus "`Masks`" will be chosen to `Show`. Note that the `Analyze Particles` function works on binary images

[2] Circularity measures how round, or circular-shape like, the object is. In Fiji, the range of this parameter is between 0 and 1. The more roundish the object, the closer to 1 the circularity.

[3] Compactness is a property that measures how bounded all points in the object are, e.g. within some fixed distance of each other, surface-area to volume ratio. In Fiji, we can find such measurements options from the downloadable plugin in `Plugins -> 3D -> 3D Manager Options`.

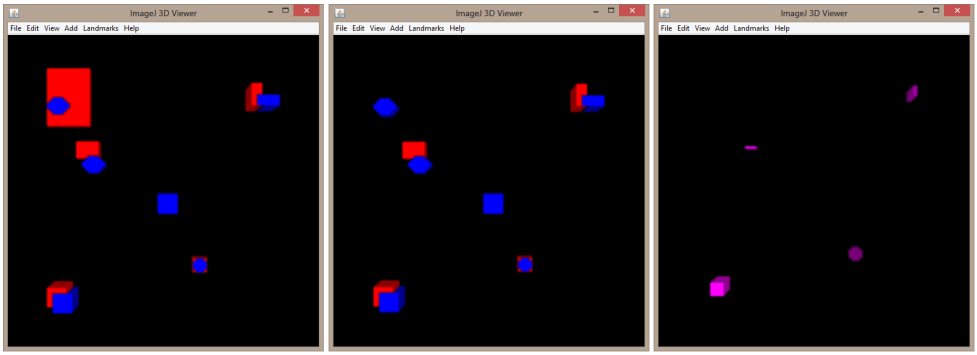


FIGURE 4.11: Synthetic 3D dataset after first filtering in 3D (*left*), then also in 2D (*middle*), and the overlapping regions after the filtering steps (*right*).

only, then we first need to convert the label image by the 3D Object Counter, which is a label image. A simple thresholding should work, since the label starts at 1. Normally the binary image after thresholding has value 0 and 255. Sometimes, a binary image of value 0 and 1 makes further analysis easier. To do so, we could divide every pixel by this image's non-zero value, i.e. 255, using [`Process -> Math -> Divide`].

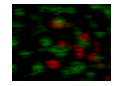
Question: What should we do if the 2D filtering is to be done in the XZ plane?

If the axes are swapped, so as the original XZ plane will be the new XY plane, then we could apply the same procedure on the axes-swapped image. This can be done with e.g. [`Plugins -> Transform -> TransformJ -> TransformJ Rotate`], which applies to the image a rotation transform around the three main axes. Please note that if the image to be processed is large, this will not be the optimal solution as it will be both time and computation expensive to apply a transform and interpolate the whole image. A faster option could be [`Plugins -> Transform -> TransformJ -> TransformJ Turn`]. The advantage of using this plugin rather than the more generally applicable [`Plugins -> Transform -> TransformJ -> TransformJ Rotate`] is that it does not involve interpolation of image values but merely a reordering of the image elements. As a result, it generally requires much less computation time, and does not affect image quality.

Option #2: Another possibility, probably the easier option in many cases, is the erosion + dilation operations in 3D. In this case, we would only want to erode and then dilate in one dimension - the one where object-to-be-removed is thin. Let's try it, `Process -> Filters -> Minimum 3D` and then `Process -> Filters -> Maximum 3D` with the same radius settings in each dimension. We will set both X and Y radius to 0 and try with Z radius being 3.0, because the large object in Channel 1 is thin in Z axis. In general it should work - provided the filter radius is not smaller than the object radius along its smallest dimension, then the object should disappear and not return. This also gives the possibility to filter anisotropically considering the fact that in many microscopy

images the Z spacing is larger than pixel size in then XY plane. Note that the erode/dilate alternative in the `Plugins -> Process` submenu will not work, because the filter size is not adjustable). On the other hand, please keep in mind that the erode and dilate operations may modify object shape, especially when objects are not roundish blob-like.

Depending on specific applications, more filtering steps can be applied before or after the previous size filtering, such as shape or location. In this session we will not discuss in details and assume the size filtering is sufficient for our task (Fig. 4.11 *middle*).



4.4.2 Finding spatial overlapping objects in both channels

In order to find colocalized objects, we will first find the overlapping (or shared) parts of the two filtered channels, and then identify the corresponding objects in each channel that contain these overlapping regions. To achieve this, what do we need to do? There are multiple ways, all of which involve:

- in each channel, label the objects so as to identify them;
- in each channel, calculate the volume (in voxels) of each object;
- compute the objects' overlapping regions of the two channels;
- calculate the volume of each overlapping region.
- find the labels of objects in each channel that overlap with some object in the other channel

These tasks could be done with the help of [`Analyze -> 3D Object Counter`] and [`Plugins -> 3D -> 3D Manager`].

We could see that in both channels, the overlapping region has value higher than zero; while the rest of the two images have either one or both channels with zero background. Therefore if we multiply the two images, only the overlapping regions will show values higher than zero. So we can run [`Process -> Image Calculator`], set the object maps of from the two channels as `Image 1` and `Image 2`, and set `Multiply` as `Operation`. Let's call the obtained multiplicative image as "`SharedMask`". Fig. 4.11 *right* shows the 3D view of the overlapping regions after the filtering steps. Note that the images of both channels that contain objects filtered by size are binary images of values 0 and 1, thus the "SharedMask" is also a binary image of values 0 and 1.

Now, if we add the "SharedMask" to the filtered binary images of each channel, the two resultant images would give background zero value, all the objects in each channel value 1, except where the overlaps is, i.e. 2. Then when using "3D hysteresis thresholding" plugin, only regions with value >= a low threshold (i.e. 1) that also contain value >= a high threshold (i.e. 2) are extracted, i.e. the objects containing the overlaps. Therefore, we have obtained also one image per channel, which contains only objects that overlap with some object in the other channel.

Since we define the object volume overlap ratio as the colocalization criteria, objects and their volume values in both channels and the SharedMask should be calculated. We need to make sure in [`Analyze -> 3D OC Options`], to check the option of "`Nb`

of Obj. voxels" (if we count voxels) or "Volume" (number of voxels multiplying voxel size). If the voxel size is not set, by default it is 1 for each dimension, thus these two parameters give the same value. After running the 3D Objects Counter, the resultant Object map gives each object a unique non-zero label and the background the zero label. Also, the measurements will be shown in a table. You may recall that in 3D OC Options menu, if "Store results within a table named after the image (macro friendly)" is not checked, the results are stored in the Results table. This way we could use the built-in macro functions nResults and getResult to access items in the table. Let's do this again, with the following code:

```
ObjVolume_ch1 = newArray(nResults);
print("ObjVolume_ch1 ("+nResults+"): ");
for(i=0;i<nResults;i++){
  ObjVolume_ch1[i] = getResult("Nb of obj. voxels", i);
  print(ObjVolume_ch1[i]);
}
```

:

This code first creates a new array, *ObjVolume_ch1*, so as to store all objects' volumes in the current channel. It is then filled with the values obtained from the Results table. For checking the code we could also print the array. Similar arrays and object labels should be created for the other channel and the SharedMask.

So far we have obtained the objects' labels and volume values in each channel and the "SharedMask" image. The next task will be to identify the labels of each object pair that overlap, in order to further find out their corresponding overlap volume ratio. Before jumping into the next part, let's ask ourselves a question - does our problem involve analyzing individual colocalized objects, or rather a global measure that tells something about colocalized objects as a whole? If the answer to your problem is the later, then we could skip the remaining of this section and the following one. Instead, probably some nice tools could do the job for us. For example as mentioned previously, the ImageJ plugin JACoP [1] offers measures, e.g. Mander's Coefficients, Overlap Coefficient, and object-based methods. We will not provide detailed descriptions on how to work with them here, please refer to their corresponding online documentation.

If you are interested in studying individual colocalized objects and their further characteristics, such as their spatial distribution, shape, size, etc, we will go on to the next task of identifying in each channel which objects overlap with objects in the other channel. Since we know the overlapping regions, then we just need to look at the same regions in each of the two channels to get these objects that have overlapping parts in the other channel. We have used ROI Manager before, now we will use another function from the plugins we installed: [Plugins -> 3D -> 3D Manager]. It plays a similar role as the ROI Manager but in 3D. So the first thing is to add the 3D overlapping regions into the 3D Manager so that we could use them for further measurements, and also for inspecting the same regions on other images such as the label images. To do so, we will use the following code[4]:

[4] Note that the "Ext" is a built-in function added by plugins using the MacroExtension interface.

```
selectImage("Objects map of Shared Mask");
run("3D Manager");
Ext.Manager3D_AddImage();
Ext.Manager3D_SelectAll();
```

:

After adding the 3D overlapping regions into the 3D Manager, we will select the object label image of channel 2 so as to find out the corresponding label values for every overlapping region. Similar to what we have done before, we can measure the maximum intensity value of each region from the label image in channel 2. So we will check `3D Manager Options` and select the `Maximum gray value`. And then click the `Quantif_3D` in the `3D Manager` window. It will give a window named "3D quantif", with a column named as "Max". This column stores the maximum gray values of each region in 3D Manager from the label image in channel 2. Since it is not the usual "`Results`" table, we can't use the `nResults` and `getResult` built-in functions to access the contents of this table. How do we obtain the objects in each channel that have these overlapping regions? There are many ways to achieve this, but let's see how to extract information from any table, in this case a table named "`3D quantif`". **The following code shows an example of how we can get one column's items from a table that is not the Fiji default `Results` table**:

```
shareObjLabelInCh2 = newArray(numSharedObjects);
selectWindow("3D quantif");
tbl = getInfo("window.contents");
tblLines = split(tbl, "\n");
idx=0;
for (i=1; i<= numSharedObjects; i++){
  lineA = split(strA[i], "\t");
  shareObjLabelInCh2[idx]=lineA[2];
}
```

:

where (in lines 3-4) "*tbl*" stores everything in the "`3D quantif`" table as *string*, and we can get each item from the table by two "`split`" operations: first into lines through the line break "`\n`" (as in line 4) and then into separate items through the tab or column break "`\t`" (as in line 7). *shareObjLabelInCh2* is an array of size *numSharedObjects*, the number of overlapping regions, which stores the object labels that contain the corresponding overlap part in channel 2. Of course, there is now a built-in function `IJ.renameResults` available (if you have ImageJ version 1.46 or later) that changes the title of a results table from one to another. Thus, we can always change the name of any table to "Results" and then use the easier built-in functions `nResults` and `getResults` to get table contents.

4.4.3 Filtering the colocalization objects by volume overlap percentage criteria

We have finally arrived to the stage that we are ready to select "real" colocalized objects from the candidates. As we will use the volume overlap criteria, let's first calculate the

volume overlap ratio. There may be several ways to define the ratio, we will use e.g. the ratio between the overlapping region and the volume of the smaller of the two objects. In order to not complicate the problem too much, we will assume that objects in one of the channels have smaller size. This could be a reasonable assumption in many biological applications. The following code realizes the process of determining the colocalization using such selection criteria, assuming the channel with smaller objects is channel 2. A new image stack, "SharedMask Filtered", is created and filled with the overlapping regions that have volume size larger than the specified percent of the corresponding object in channel 2:

```
selectImage("Objects map of Shared Mask");
run("3D Manager");
Ext.Manager3D_AddImage();
newImage("SharedMask Filtered", "8-bit black", width, height, slices);
for (j=0; j<numSharedObjects; j++){
  objLabelInC2 = shareObjLabelInCh2[j];
  voRatio=ObjVolume_shared[j]/ObjVolume_ch2[objLabelInC2-1];
  print("volume overlap ratio of "+j+"th object in channel 2 is: "+voRatio+"
     = "+ObjVolume_shared[j]+"/"+ObjVolume_ch2[objLabelInC2-1]);

  //select the objects that have volume overlapping higher than a user
    specified ratio, ``volOverlap"
  if (voRatio>volOverlap){
    numColocObjects=numColocObjects+1;
    Ext.Manager3D_Select(j);
    Ext.Manager3D_FillStack(1,1,1);
  }
}
```

:

where "numColocObjects" gives the total number of colocalization object pairs, "voRatio" computes the volume overlap ratio, and "volOverlap" is a user-specified threshold that discards objects with lower overlap ratio. `Manager3D_FillStack` fills the newly created image with the selected 3D region. For each region, the values filled can be all the same, or a different one as label. The final colocalized objects in each channel can be obtained using again the `3D Hysteresis Thresholding`, as we have done previously, but with the newly created overlapping regions image, "SharedMask Filter". In the synthetic example, the four candidate objects have volume overlap ratio of: 0.11, 0.51, 0.25, and 1 (see Fig. 4.12 *left*). And if we specify "volOverlap" to be, e.g. 0.2, 0.3, 0.52, the final "real" colocalization results differ, as shown in the three images on the right side, respectively, in Fig. 4.12.

4.4.4 Visualizing results (optional)

To better examine 3D data, Fiji offers [`Plugins -> 3D Viewer`] to visualize rendered volumes, surfaces, or orthogonal slices. After opening the "3D Viewer" window, images can be loaded using either [`File -> Open`] (for any image on disk) or [`Add -> From Image`] (for images already loaded in Fiji). Multiple images can be loaded. For overlapping images, we could modify image's transparency by [`Edit -> Change transparency`].

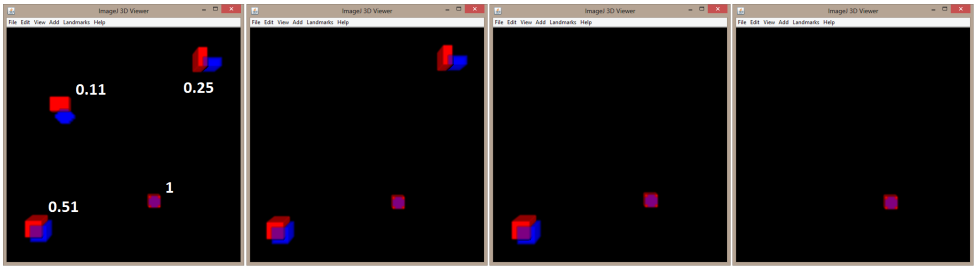


FIGURE 4.12: (*left*) Candidate colocalization objects from the two channels, the number next to each object pair shows the volume overlap ratio compared to the blue channel objects. (*middle-left - right*) Determined colocalization object pairs using specified ratio threshold of: 0.2, 0.3, and 0.52, respectively.

Image brightness can be changed using [`Edit -> Transfer Function -> RGBA`]. There are many more possibilities to control and edit the visualization properties, we will leave this as a homework for you to exploit further. Examples of visualizing 3D data using this viewer can be found in many figures in this session such as Fig. 4.11, 4.12, 4.14.

4.4.5 Testing the macro on Hela cells with viral replication

The pipeline of operations we came up with can now be assembled to a complete macro to process the original Hela cells stacks.

Segmenting spots. There are multiple tools for detecting spot-like structures, including the two methods we have used in the spots counting session. Alternatively, a learning based method is also suitable, where we can train a classifier that can "pick" objects of interest out of other structures in the same image. A popular free software tool is ilastik (`www.ilastik.org`). In order to better segment images, some preprocessing steps can be applied. A typical step is deconvolution and background subtraction. In Fiji, you can find several plugins for these tasks to try on your images, such as:

- Parallel Iterative Deconvolution (`fiji.sc/Parallel_Iterative_Deconvolution`), where the point spread function (PSF) can be estimated using the Diffraction PSF 3D plugin (`fiji.sc/Diffraction_PSF_3D`) (for our images, you can try to select "WPL" as Method, and use 20 or more iterations; PSF estimation needs media refractive index, numerical aperture, wavelength and pixel spacing. You can also find "PSF_C1.tif" and "PSF_C3.tif" for the two channels, estimated from the Diffraction PSF 3D plugin.) An example can be found in Fig. 4.13.

- To further remove background noise, you can try [`Process -> Subtract Background`] (for our images, the rolling ball radius can be set to 10 pixels)

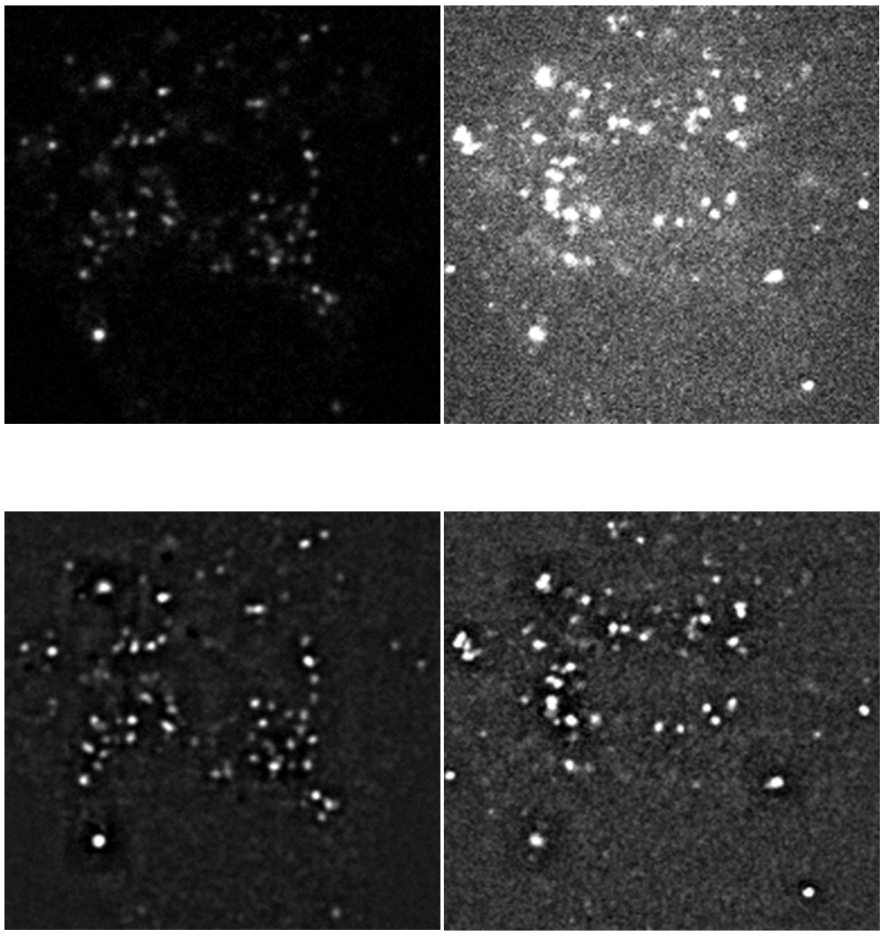


FIGURE 4.13: Example images before (*top*) and after (*bottom*) deconvolution.

So now when we try to analyze the Hela cell images, the parameters used for the synthetic dataset may not be applicable anymore. Of course we could manually modify each value through the entire macro file we made. But this is not efficient. So it is better to use variables for these parameters, and specify them e.g. in the beginning of the macro code.

After the macro is “parameterized", we can specify values. For this dataset, let’s set the threshold to be 0.5 for both channels, and 3D object sizes to be [5, 500], and maximum 2D object sizes in XY plane are 150 (for channel 1) and 50 (for channel 2). And volume overlap

ratio threshold can be any value between 0 and 1. Fig. 4.14 shows a few example results of the Hela cell images.

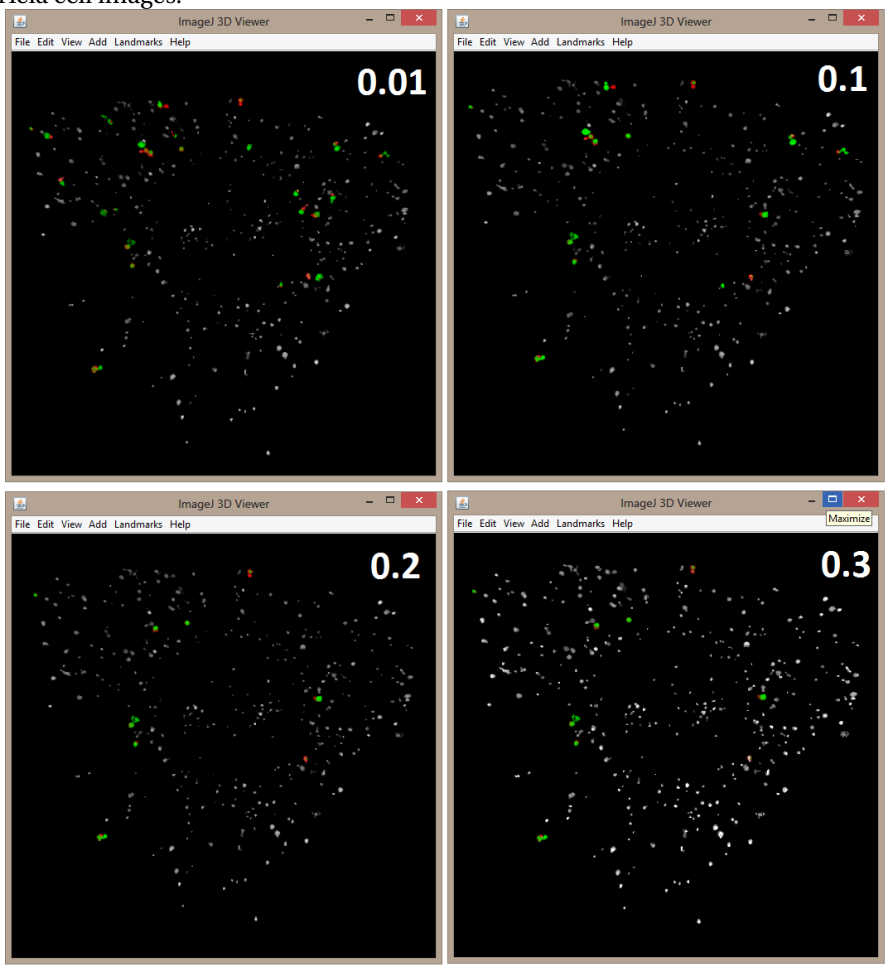


FIGURE 4.14: Colocalized objects in channel 1 (*red*) and channel 2 (*green*), together with all filtered objects in channel 1 (*white with transparency*) using the specified ratio thresholds: 0.01, 0.1, 0.2, and 0.3. Their corresponding number of determined colocalization objects are: 25, 14, 9, and 9, respectively.

Please note that for a complete colocalization analysis, further examination steps are needed such as accuracy evaluation, robustness evaluation, and reliability evaluation like comparing to random events. These are out of the scope of this session thus not discussed here.

4.4.6 Setting up a parameter interface (optional)

You may find modifying parameters inside the source code of the macro not an elegant usage. If you are willing, constructing a simple user interface window for parameters is made very easy in Fiji. The `Dialog.*` functions offers a practical dialog box/window with just a few lines of code. An example is shown in Fig. 4.15, with the code on the left side and the generated dialog window the right side. Please note that the parameters that fetch the values from the dialog should be specified following the same top-down order as the dialog. Now we can create customized dialog window for the parameters that we would like user to specify.

```
Dialog.create("Dialog Window Title");
Dialog.addNumber("Probability map threshold: ", 0.5);
Dialog.addMessage("");
Dialog.addNumber("Minimum 3D object size (in voxels): ", 5);
Dialog.addNumber("Volume overlap for colocalization: ", 0.2);
Dialog.addMessage(" ");
Dialog.addCheckbox("Display in 3D Viewer?", false);
Dialog.show();

thres_pm_ch1 = Dialog.getNumber();
min3DObjSize=Dialog.getNumber();
volOverlap = Dialog.getNumber();
Display3D = Dialog.getCheckbox();
```

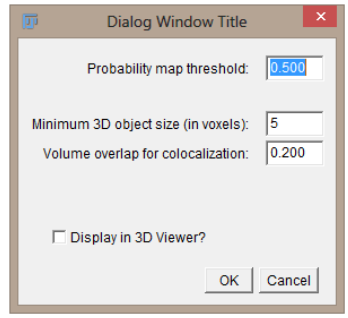


FIGURE 4.15: An example of creating a dialog window for parameters to be specified by users.

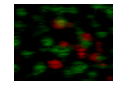
In case you do not have time to write up the complete macro during the session, a possible solution is provided in your Session5 folder (`colocalization.ijm`).

4.5 Tips and tools

- ImageJ Built-in Macro Functions: http://rsbweb.nih.gov/ij/developer/macro/functions.html
- 3D ImageJ Suite: download: http://imagejdocu.tudor.lu/lib/exe/fetch.php?media=plugin:stacks:3d_ij_suite:mcib3d-suite.zip and unzipping it in your plugins folder. Detailed description of the plugin: http://imagejdocu.tudor.lu/doku.php?id=plugin:stacks:3d_ij_suite:start.
- JACoP (includes object-based method): http://imagejdocu.tudor.lu/doku.php?id=plugin:analysis:jacop_2.0:just_another_colocalization_plugin:start
- Math-Clinic Short Tutorials: http://cellnetmcweb.bioquant.uni-heidelberg.de/index.php/Short_Tutorials

4.6 Groups close by which work on similar problems

- Dr. Holger Erfle's group, ViroQuant (http://www.bioquant.uni-heidelberg.de/technology-platforms/viroquant-cellnetworks-rnai-screening-facility/contact.html), located at Bioquant
- Dr. Karl Rohr's group, Biomedical Computer Vision Group (http://www.bioquant.uni-heidelberg.de/?id=322), located at Bioquant



4.7 References

[1] S Bolte and F P Cordelières. A guided tour into subcellular colocalization analysis in light microscopy. *Journal of Microscopy*, 224:213–232, 2006.

[2] K W Dunn, M M Kamocka, and J H McDonald. A practical guide to evaluating colocalization in biological microscopy. *Am J Physiol Cell Physiol*, 300:C723–C742, 2011.

[3] P A Fletcher, D R Scriven, M N Schulson, and D W Moore. Multi-Image Colocalization and Its Statistical Significance. *Biophysical Journal*, 99:1996–2005, 2010.

[4] T Lagache, V Meas-Yedid, and J C Olivo-Marin. A statistical analysis of spatial colocalization using Ripley's K function. In *ISBI*, pages 896–899, 2013.

[5] B Obara, A Jabeen, N Fernandez, , and P P Laissue. A novel method for quantified, superresolved, three-dimensional colocalisation of isotropic, fluorescent particles. *Histochemistry and Cell Biology*, 139(3):391–402, 2013.

[6] A Rizk, G Paul, P Incardona, M Bugarski, M Mansouri, A Niemann, U Ziegler, P Berger, and I F Sbalzarini. Segmentation and quantification of subcellular structures in fluorescence microscopy images using Squassh. *Nature Protocols*, 9(3):586–596, 2014.

[7] S Wörz, P Sander, M PfannmÃűller, R J Rieker, S Joos, G Mechtersheimer, P Boukamp, P Lichter, and K Rohr. 3D geometry-based quantification of colocalizations in multichannel 3D microscopy images of human soft tissue tumors. *IEEE Transactions on Medical Imaging*, 29(8):1474–1484, 2010.

[8] V Zinchuk and O Zinchuk. Quantitative colocalization analysis of confocal fluorescence microscopy images. *Curr. Protoc. Cell Biol.*, U4:19, 2008.

5

RELATED RESOURCES

To list just a few BioImage Analysis related tutorials, slides, online documentations, online books...

- Fiji online documentation: http://fiji.sc/Documentation
- ImageJ User Guide: http://rsbweb.nih.gov/ij/docs/guide/146.html
- EuBIAS 2013 - BioImage Data Analysis Course: http://eubias2013.irbbarcelona.org/bias-2-course
- CMCI ImageJ Course page: http://cmci.embl.de/documents/ijcourses
- CMCI Macro programming in ImageJ Textbook: https://github.com/cmci/ij_textbook2/blob/master/CMCImacrocourse.pdf?raw=true
- ImageJ macro programming: http://www.matdat.life.ku.dk/ia/sessions/session12-4up.pdf
- Fluorescence image analysis introduction:http://blogs.qub.ac.uk/ccbg/fluorescence-image-analysis-intro/
- 3D/4D image reconstruction using ImageJ/Fiji: http://www.encite.org/html/img/pool/7_3D_image_reconstructions_p115-144.pdf
- Math-Clinic BioImage Analysis events page: http://cellnetmcweb.bioquant.uni-heidelberg.de/index.php/BioImage_Analysis
- BioImage Information Index: http://biii.info/
- ...

Notes:

Notes:

Notes:

Notes: