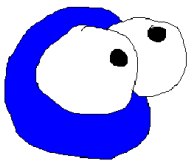
# Hcell

**High Content Screening Image Analysis software.**



## Hcell in 3 Steps!!

1. **Go to the website and create your pipeline.xml file.**

2. **Download the application and install it.**

3. **Open a command prompt and type**

```
hcell -i input_directory -o output_directory -f pipeline.xml
```

*Manual created by José Miguel Serra Lleti.*
*Any questions contact by serrajosemi@gmail.com*

# PREFACE

Hcell is an open-source software command line tool for performing processing and segmentation in high content screening datasets.

This tool was created by the author of this manuscript during the summer-winter of 2013 at the Boutros Lab - Signaling and Functional genomics - in DKFZ (German Cancer Research Center, Heidelberg, Germany). In that lab they were researching (between other amazing things), gene interactions and the effects of gene perturbation in the cell phenotype.

One of the ways of studying that is using image analysis. We do our experiments, and then we compare the images coming from our perturbed cells against our controls (the cells as they are in nature). Now, if for example we measure the size of cell nuclei and we find a statistical significance that the cells have changed their nucleus size, we have found an interesting gene then! Or at least a gene that does something…

When I arrived at DKFZ they gave me a bunch of data to perform that kind of analysis. I must confess I had no idea where to start. So they told me: "If this is the first time you are doing analysis, maybe you can try some of the existent amazing user friendly image analysis tools, like Fiji (http://fiji.sc/Fiji) or CellProfiler (http://www.cellprofiler.org/ ). Or you also can play with R or Matlab."

I already knew Fiji (Fiji is Just Imagej), is a good tool when you want to do a quick and dirty preprocess or segmentation analysis for images. CellProfiler, however, was a completely new tool for me. I found it really friendly and I really recommend it for beginners, is a fantastic tool.

If we have such amazing tools, also open source, why I decided to develop Hcell? Two main reasons: first, the number of segmentation algorithms for cell image analysis are quite scarce. They argued that more complex and advanced segmentation algorithms, were too slow or unnecessary (in statistical terms) when applied to large datasets. I completely disagreed. I liked ITK (http://www.itk.org/) and I thought, why don´t include level sets or registration methods inside the pipelines and try to get better quantitative data?

The second reason was more personal. I started to hate some of the programs that I was testing (I will be polite and I will not mention which ones specially) because they were extremely slow. I didn´t have the patience to wait two days for my data to be completely processed. And then, if a mistake was done in my pipeline (even a small one), again I needed to recompute everything. That was too much for me.

One day a friend showed me an application with OpenCV and I thought it was amazing. Why don´t integrate OpenCV, ITK and maybe other fancy libraries in one program? I could grab one function from library A, other from library B and integrate them in a unique workflow, completely suited to my needs… and fast!

**Who should use Hcell?**

1) If you are biologist analyzing data and working with large datasets, ranging the order of Terabytes.
2) If you are a developer and you can program C++, you can develop a module and integrate it into your pipeline. For example, you can easily integrate GPU based modules or include any C++ library, compile your code and run it.
3) If you don´t like to wait several days to see your results, you can compute your dataset in your own laptop!

**Why Hcell is faster?**

The first reason, it uses a simple principle: use just what you need. Hcell does a precheck and if everything is ok, then is just a pool with your images and a list of calls to the functions of interest being executed one after another. This means that the code is highly efficient and input errors are discovered at a sort of "simulated" compile time, rather than at program execution. It is completely decoupled from a GUI, and uses the minimal memory needed by your system.

The second and main reason is **OpenCV** (*Open Source Computer Vision Library*). OpenCV is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez (*source:Wikipedia*). OpenCV is released under a BSD license and hence it's free for both academic and commercial use.

**I tried Hcell and I think Hcell needs more functions. Why is that?**

You cannot compare (so far) Hcell with the existent programs I commented above. Those programs have been developed during many years. Hcell is young, but I hope is starting to born. The possibilities are incredible (GPU, parallelization support, image web server, faster and newer algorithms…) and I would like to encourage people to submit their own code and add it to the program in the new coming versions. The authorship will be always respected and your function will be added to the website! Please share your code!

I wish we could develop new ideas together in a good community environment and lead Hcell to the next level!

J.M. Serra

# ACKNOWLEDGMENTS

German Cancer Research Center Heidelberg
Division B110
Signalling and Functional Genomics

*Hcell would not have been possible without the following libraries:*

Such a great library! http://opencv.org/

Our input XML file is read using a TinyXML. http://www.grinninglizard.com/tinyxml/

Some of the implemented functions are based on the Bioconductor R package EBImage:

http://www.bioconductor.org/packages/2.12/bioc/html/EBImage.html

Notepad++ has been used to visualize the xml pipelines.

# Chapter 1: Getting started with Hcell

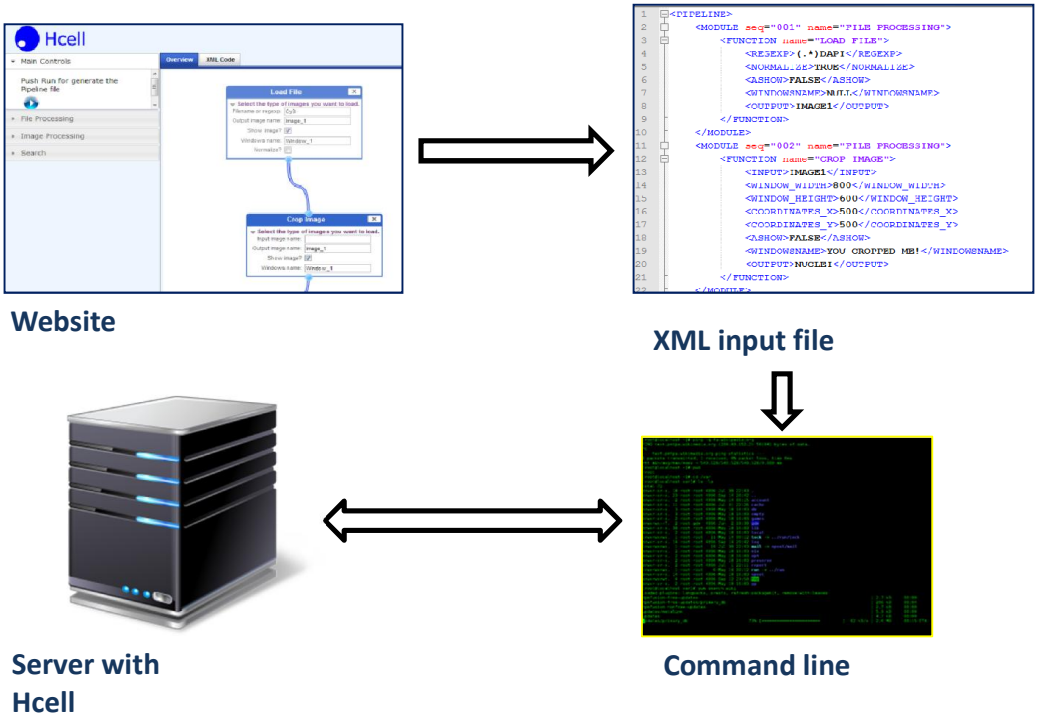
## 1.1 Introduction

High content morphology screening is a type of phenotypic screening method used in biological research to identify how perturbations by specific substances, such as RNA interference, alters the visual appearance of a cell. It is referred to as "high content" as it involves measuring and analyzing several parameters of the observed phenotype. Typical analysis of these screens involves automatic segmentation of fluorescent images and then, determining, for example, where the edges of an object are, counting similar objects, calculating the area, perimeter length and other useful measurements of our regions of interest inside the digital image. **Hcell** has been created for this purpose.

**Hcell** is a command line tool which gives the user high flexibility in developing new cellular assay analysis routines in an automated software way.

A selection of segmentation algorithms, pre-processing and post –processing tools is provided. **Hcell** can also been used as an image processing tool without the analysis. The pipeline is in fact a program formed by a set of functions, which can be used in the same way as a common library in programming.

The basic method to construct a user-defined analysis pipeline is described in detail in the following points.




The idea behind Hcell is to keep the program as simple as a program it could be. After the image acquisition, once installed, Hcell can be called from the command line as follows:

```
user$ hcell –i input_directory –o output_directory –f pipeline1.xml
```

Where:

**-i input_directory :** input directory where the images are directly located. Current version doesn´t read subdirectories.

**-o output_directory :** output directory where the files resulting from feature extraction or the images saved are stored. If the directory doesn´t exist, it will be created.

**-f pipeline1.xml :** XML file with the pipeline workflow. The XML file which describes the processing pipeline can be created using our website tool or manually, adding XML functions with a simple notepad.

**-- verbose:** Provides extra info about inner operations performed during the analysis. Optional.

NOTE: Directories are referred in the UNIX style, with the forward bar / . Example:

```
C:/my_plate/my_pictures/
```

## 1.2 The XML pipeline

The common pipeline is usually set up following the next steps:

- **Object and target selection**. What we want to measure or locate? Typical targets are cell nuclei, bodies or a specific protein distribution inside the body. The first step is load the images from the different channels in the right order, and having an image of the workflow we want to perform.
- **Preprocessing**. Preprocessing operations are applied before segmentation in order to remove noise, artifacts or delimiting the area of application. Each preprocessing function has adjustable parameters that can be adjusted to suit a particular user need.
- **Segmentation**. Segmentation algorithms are designed to extract the pixels of our interest, and these objects can be separated and isolated as a group of objects for further analysis. Some algorithms have adjustable parameters which should be tuned by the user in order to optimize the segmentation process.
- **Postprocessing**. Sometimes image processing algorithms can optionally be applied to a given image after segmentation for further refinements.
- **Feature extraction**. The last step is specify which features we want to measure from the objects: gray level intensity, number of objects, area, length, diameter… Hcell doesn´t perform statistical analysis, but provides a file in a tab format which can be easily imported in other programs like Matlab, R or Excel.

Once the main ideas of the workflow are stablished and we know more or less what are we going to do, we need to create a XML file calling the proper functions. One way of doing this, is visiting the website www.hcellapp.blogspot.com and connect with the main application. The HCell XML editor webpage will help you to obtain a XML file that suits your needs. Otherwise, you can create the XML using a simple text editor.

The XML functions gave complete control over the sequence of steps. The XML is really simple, starts with the tag <PIPELINE>, followed by a module tag <MODULE> and a function tag <FUNCTION>. Modules are generalized sets of functions. Specifically, the modules are:

- **FILE PROCESSING**
  All functions related to managing input and output or visualization processes.
- **IMAGE PROCESSING**
  Post-processing image operators, like morphological or gradient operators, filters and brushes.
- **SEGMENTATION PROCESSING**
  Segmentation algorithms and other special functions, for example, for tagging objects.
- **COMPUTE FEATURES**
  Feature extraction from images, like mean intensity, moments or haralick texture features. A detailed description will be given later.

Here the typical function is shown:

```
<MODULE seq="013" name="FILE PROCESSING">   Module tag

        <FUNCTION name="NORMALIZE">    Function tag with function name

            <INPUT>IMCROP</INPUT>      Parameters
            <OUTPUT>BODIES</OUTPUT>
            <ASHOW>TRUE</ASHOW>
            <WINDOWSNAME>NULL</WINDOWSNAME>
            <!-- MAXMIN or LOG-->   Comment Tag
            <TYPE>LOG</TYPE>

      </FUNCTION>

</MODULE>
```

A Module can contain several functions only if they belong to the same module type.

XML tags are always capital letters and formed by a unique word. XML attributes, like name, are always strings enclosed in quotation marks, and can have spaces, but they are also in capital letters.

In case the Module, Function or any parameter tag is not properly written, an error message will be shown indicating the module where the error was found.

The **seq** attribute on each module is a string that can be used as a reference for the user. The attribute **seq** is not mandatory, and it is not taken on account during the execution. It helps to locate the module in case of error and to navigate inside the XML pipeline.

Any XML structure methods are usually shown by console. Semantics (proper meaning) of parameters is also checked but with limitations.

In case you find an unexpected error not shown in console, or extra difficulties, please send an email to serrajosemi@gmail.com with the subject HCELL PROBLEM. I will try to fix it as soon as possible!

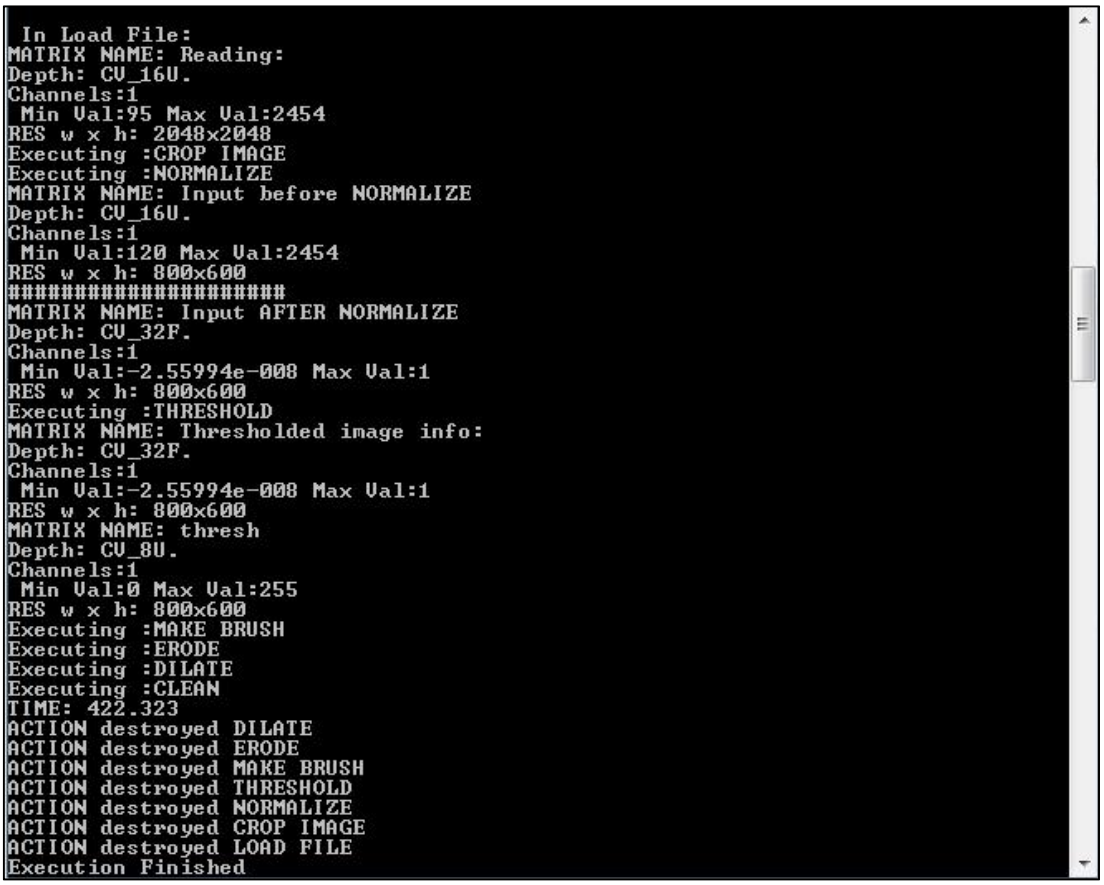
## 1.3 How to manage a collection of images.

The first operation we need to perform is to load images. **Every pipeline must start with a load statement.**

Hcell is composed of two main methods, *preparePipeline()* and *start()* . The first method gets the XML file parameters and stores them in the pipeline. This is needed for preparing loads and writes and features file saving.

When a load file instruction is set, it goes to the directory and searches any file who can match with the regular expression provided by the user. All name files are pre-loaded into a list for processing.

**Each load file instruction needs to find the SAME number of files for each loaded channel.** Loading is supposed to be done for each independent channel. If we have 1500 images from DAPI channel, and we load now a FITC channel, we will need to have also 1500 images from the corresponding FITC channel, not 1400 or 1600. Use regular expressions if you want to match a subset of images from your original set, or move them to a different directory.



In the figure, typical output (pipeline2.xml) of a program in verbose mode is shown. Information regarding images as matrices of data are displayed: Number of Channels, Minimum and Maximum Value, depth (8, 16 or 32 bits).

Use this information if you find unexpected problems.

Hcell so far only work with grey value images. RGB images can be load, but there is no guarantee that when used in a function is not going to give you an error (because the function needs a binary or an 8-bit image). If you load a color image (8 bits, 3 channels) and you try to pass it to a function where only 8 bits 1 channel is allowed, you will get an error. Verbose mode is useful to find those errors.

NOTE: Verbose mode is continuously under improvement. Some functions, can change version to version in the output message.

## 1.4 Catalog of functions

- Image IO
  - Load and write in tiff, jpeg, png, pbm, ppm.
- Image processing
  - Filters
    - Gblur
    - Laplacian
    - Sobel
    - Canny Edge
  - Median filter, blur, speckle
  - Filter 2D
  - Create Kernel
  - Morphological Operators
    - Erode, Dilate, Opening, Closing, …
  - Thresholding
    - normal, Otsu
  - Thresholding Adaptive

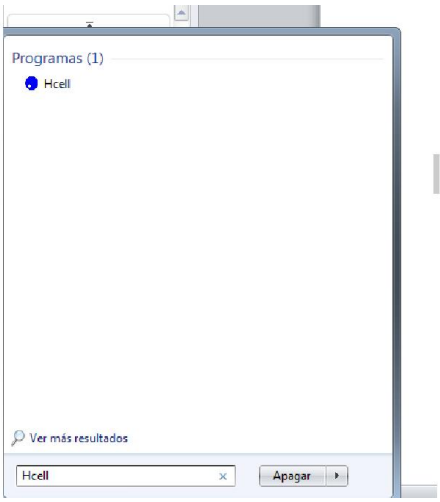
- Segmentation Processing
  - Watershed
  - Voronoi propagation
  - Nuclei clumps splitting.
    - Split by otsu, watershed, fragmentation

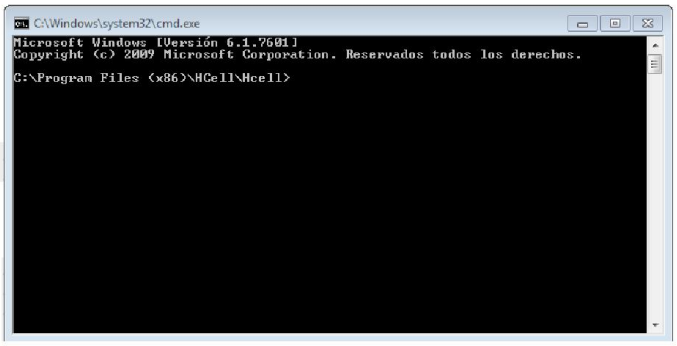
- Feature Extraction (16 major features so far)
  - Intensities (mean, sd, quantiles, mad)
  - Shape ( area, perimeter, radius mean, max and min, roundness)
  - Moments (eccentricity, major and minor axis, centroid)
  - Haralick Texture Features

# Chapter 2: Installation.

## 1.1 Windows

Download HCell or HCell64 if your machine is 64 bits. Follow the setup instructions. Once installed, click on the icon on your desktop or in the icon on programs and a command line windows will appear. This command line has by default the installation directory of the application. Now you just need to type hcell in the programs or click in the application icon. The command prompt will appear and then you can proceed as explained in chapter 1.1.




## 1.2 Unix/Linux

STEP 1: For installing Hcell in a clean installation of a Unix system, you are going to need the following libraries:

- OpenCV http://opencv.org/
- Boost library : http://www.boost.org/

There are two ways to install those libraries. The easy one, if you are using a distribution whit aptitude, just type in the command line:

```
user$ sudo apt-get install opencv
user$ sudo apt-get install libboost1.55-dev
```

Sometimes boost is not directly available using aptitude and using the command *sudo apt update* doesn´t help either. In that case you can do something like this:

```
user$ mkdir -pv /tmp/boostinst
user$ cd /tmp/boostinst/
user$ wget -c
'http://sourceforge.net/projects/boost/files/boost/1.55.0/boost_1_55_0.tar
.bz2/download'
tar xf download
cd boost_1_55_0/
```

```
./bootstrap.sh --help
./bootstrap.sh --show-libraries
./bootstrap.sh

checkinstall bjam install
```

If you experience more problems installing boost library or opencv, it is recommended to read instructions in the following links:
http://www.boost.org/doc/libs/1_41_0/more/getting_started/unix-variants.html
http://docs.opencv.org/doc/tutorials/introduction/linux_install/linux_install.html

For Ubuntu versions below 13.10 (like 12.04 LTS) it is better if you follow the instructions from here:

https://help.ubuntu.com/community/OpenCV

Here is a more specific explanation, with the basical installation only for running Hcell. To install OpenCV on the Ubuntu 12.04 operating system, first install a developer environment to build OpenCV.

```
sudo apt-get install libopencv-dev
sudo apt-get install build-essential checkinstall cmake pkg-config yasm
sudo apt-get install libtiff4-dev libjpeg-dev libjasper-dev
sudo apt-get install libtbb-dev
sudo apt-get install libgtk2.0-dev
```

Now go to the website :
**http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/**

### and download the latest unix opencv version.

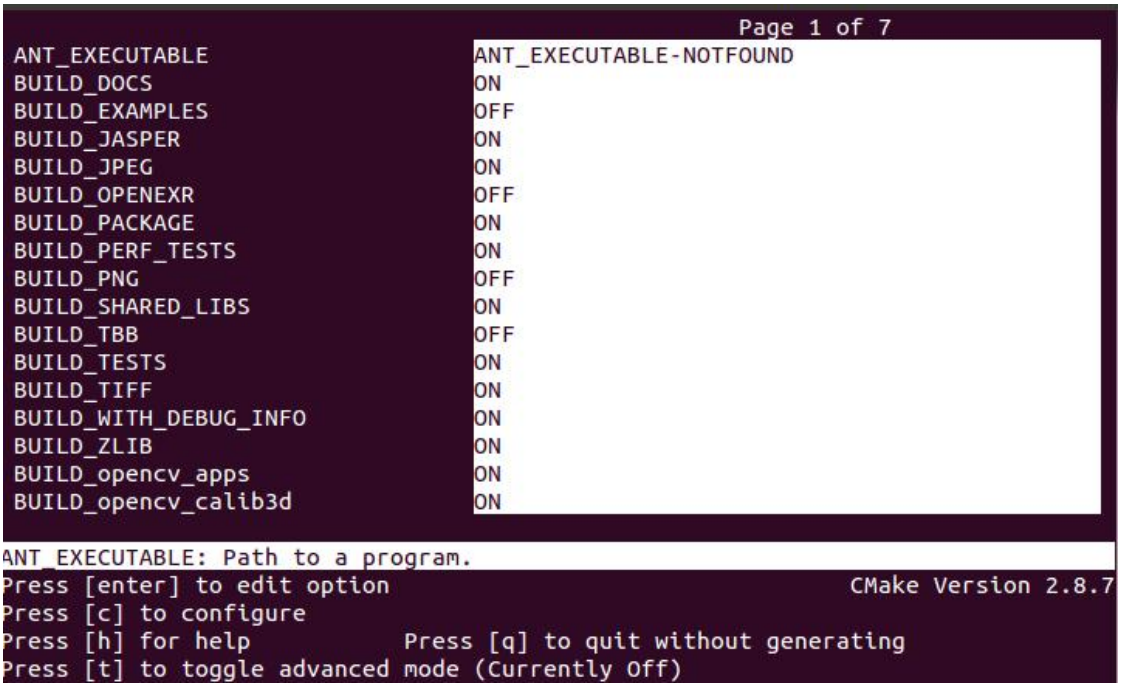
```
unzip opencv-2.8.0.zip
cd opencv-2.8.0/
mkdir build
cd build
```

or use ccmake installing via

```
sudo apt-get install cmake-curses-gui
```

go to the build directory and type

```
ccmake ../.
```




Select from the list the features you want to install. I recommend to turn on the TIFF library. Then push c (configure) and if everything is ok, g (generate). Go back to the previous directory and type:

```
cd ..
make
```

If compilation finishes without errors, you can install by saying:

```
sudo make install
```

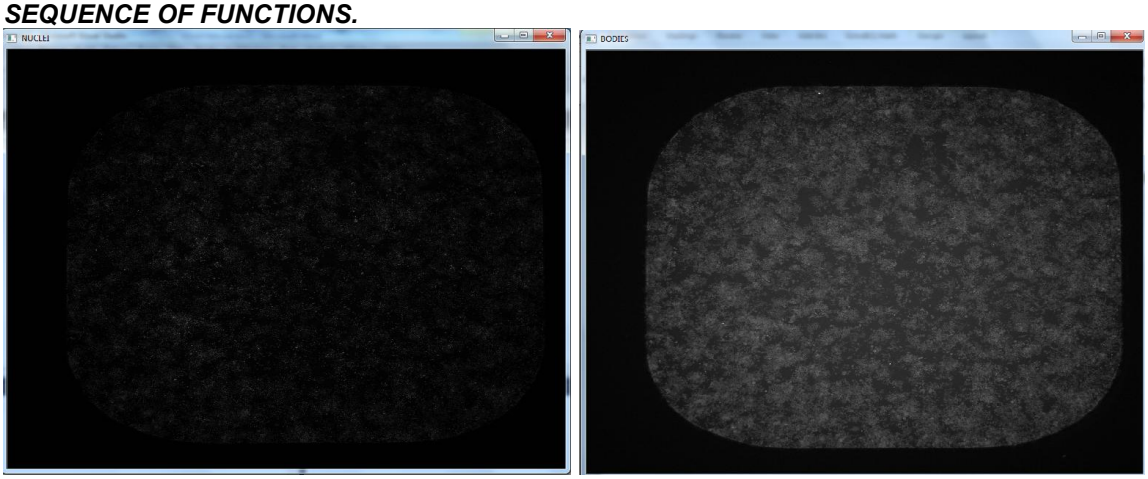
# Chapter 3: Examples with Hcell.

With Hcell a series of Examples is provided. These examples are made for playing with the program: you can modify parameters and see what happens!! In the next pages a brief description of what can you find on each example is explained. If you want to know more about how to use functions, look Chapter 5.

***EXAMPLE 1:*** `hcell -i "./images" -o "./output" -f pipeline1.xml`

***DESCRIPTION:*** Loads a DAPI and FITC images, then normalizes them, and shows them. Different types of normalization are used, maxmin for dapi, and log for fitc.

***SEQUENCE OF FUNCTIONS.***




LOAD
NORMALIZE
SHOW

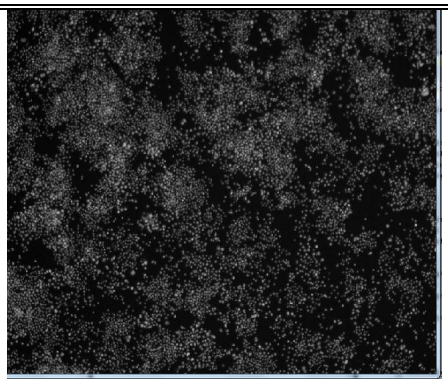
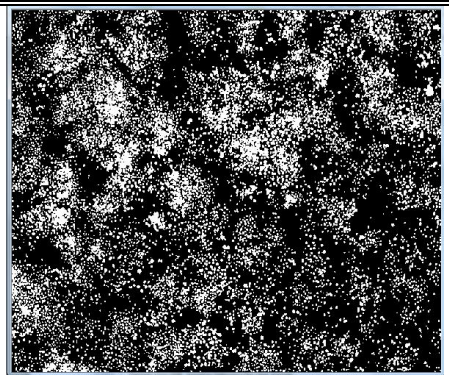
***EXAMPLE 2:*** `hcell -i "./Images" -o "./Output" -f pipeline2.xml`

***DESCRIPTION:*** Loads an image, crops a region, normalizes it and then applies a threshold. After that a brush is created for morphological operators, in this case, erode and dilate are applied.

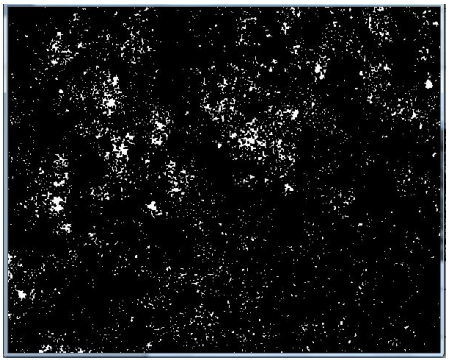
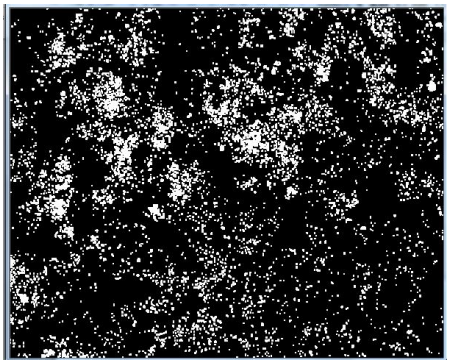
Pipeline parameters have not so much coherence, it just shows an example. Play with parameters for having an idea of how morphological operators affect binary images using different brushes.

***SEQUENCE OF FUNCTIONS.***

You should get this as result:

- LOAD
- CROP
- NORMALIZE
- THRESHOLD

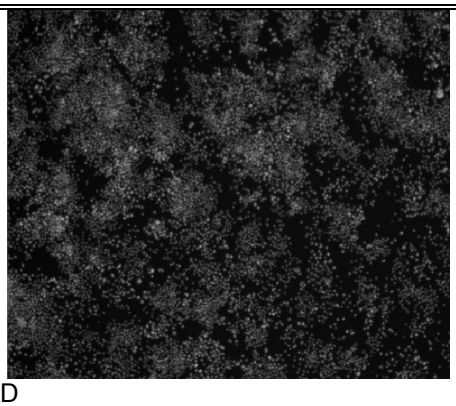
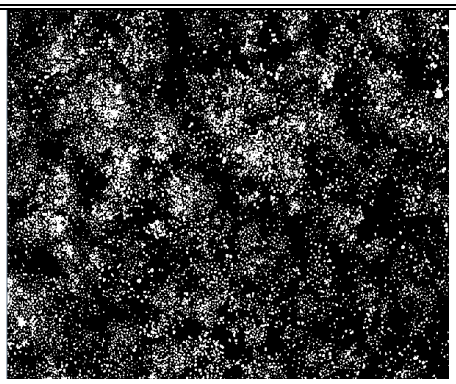
- BRUSH
- ERODE
- DILATE

***EXAMPLE 3:*** `hcell –i “./Images” –o “./Output” –f pipeline3.xml`

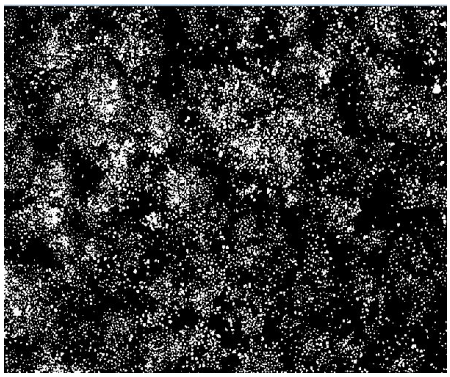
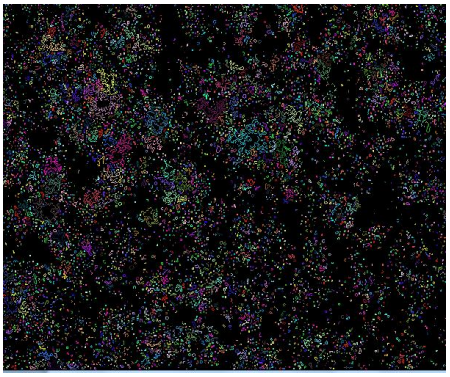
***DESCRIPTION:*** Loads a DAPI image, then normalizes it. Global thresholding is applied, followed by an opening and a function for finding contours and storing them. Final contours are saved in a PNG image.

***SEQUENCE OF FUNCTIONS.***

You should get this as result:

LOAD
CROP
NORMALIZE in Log scale

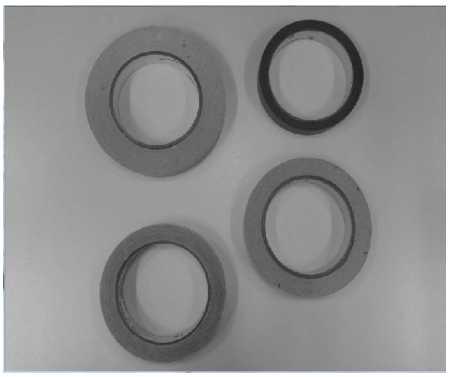
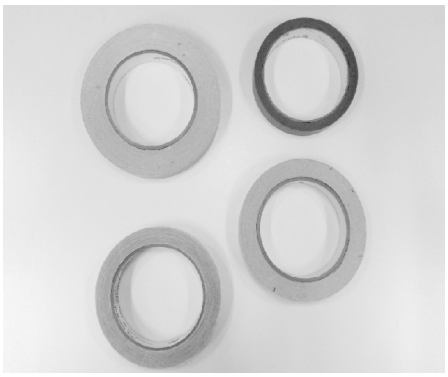



THRESHOLD global
Gaussian BRUSH and applies OPENING
FIND CONTOURS
WRITE result to a PNG as MYIMAGE_0_date.png

***EXAMPLE 4:*** `hcell –i “./Images” –o “./Output” –f pipeline4.xml`

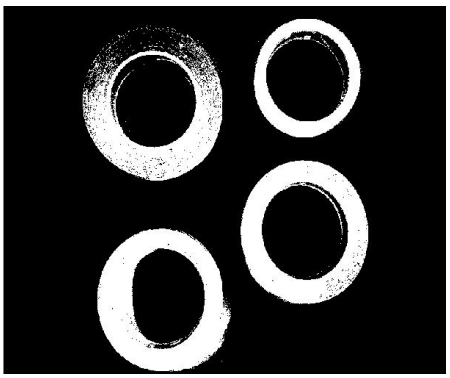
***DESCRIPTION:*** Extract canny edges from a binarized image.

***SEQUENCE OF FUNCTIONS.***

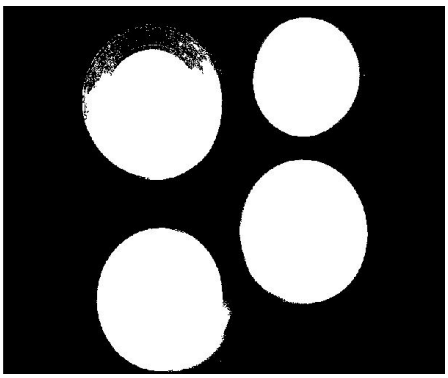



LOAD tape picture. NORMALIZE in Log scale

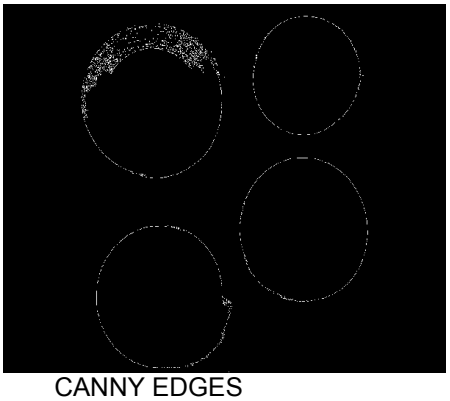
SHOW tape picture.



THRESHOLD Inverted



FILL HULL

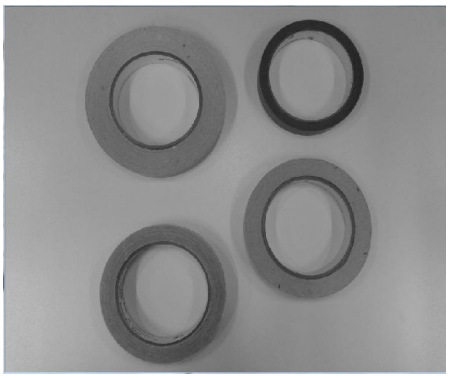


CANNY EDGES

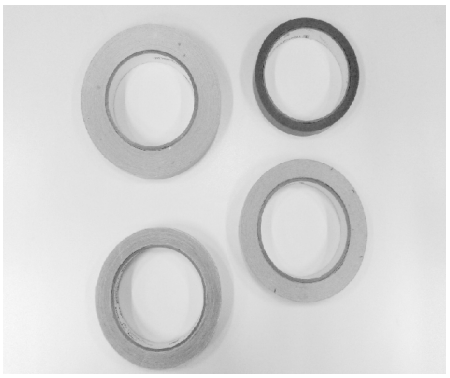
***EXAMPLE 5:*** `hcell –i “./Images” –o “./Output” –f pipeline5.xml`

***DESCRIPTION:*** Extract boundaries using a morphological operation from an inverted binarized image.

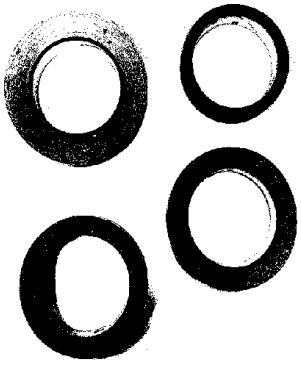
***SEQUENCE OF FUNCTIONS.***



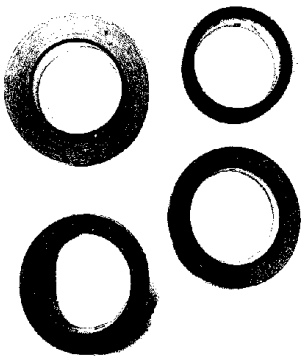
LOAD tape picture.
SHOW tape picture.



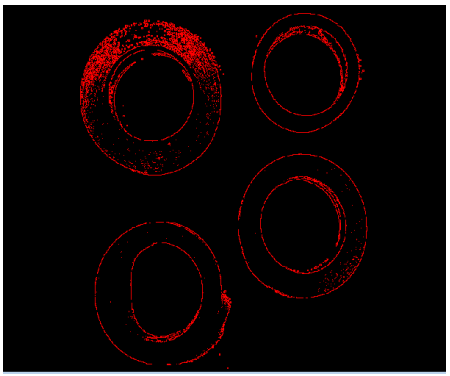
NORMALIZE in Log scale



THRESHOLD Inverted



MAKE BRUSH
ERODE



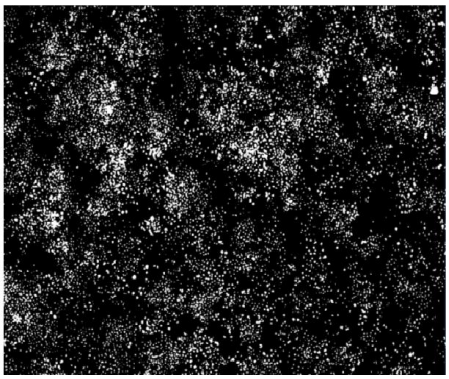
BOUNDARY EXTRACTION
With RED colour.

***EXAMPLE 6:*** `hcell -i "./Images" -o "./Output" -f pipeline6.xml`

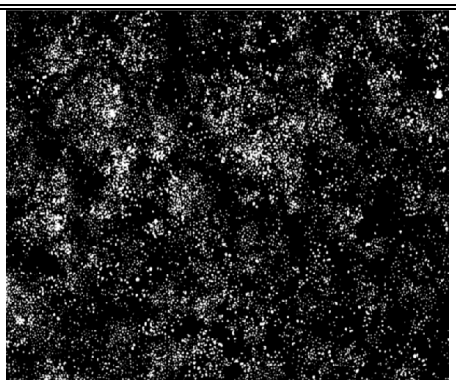
***DESCRIPTION:*** Applies Gaussian Blur and Sobel derivative operator.

***SEQUENCE OF FUNCTIONS.***

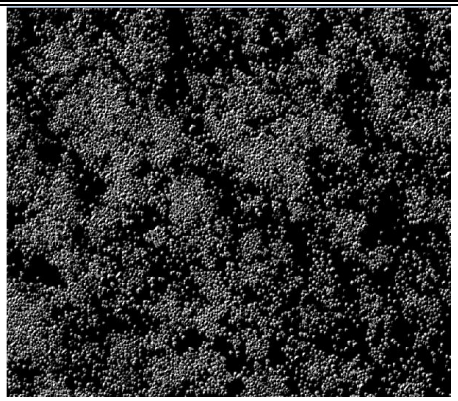
LOAD
CROP
NORMALIZE
THRESHOLD
MAKE BRUSH
As Example 3



EROSION



GAUSSIAN BLUR



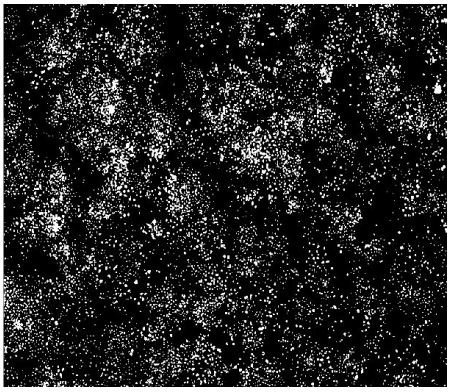
SOBEL FILTER

***EXAMPLE 7:*** `hcell –i “./Images” –o “./Output” –f pipeline7.xml`

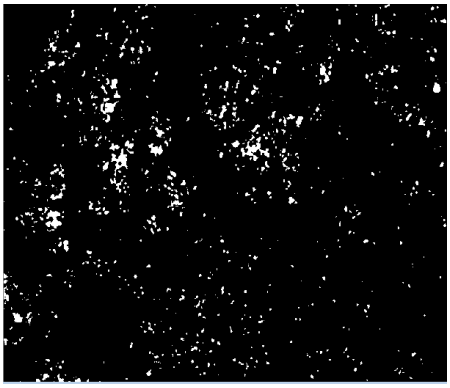
***DESCRIPTION:*** Applies a median filter after a morphological operator.

***SEQUENCE OF FUNCTIONS.***

LOAD
CROP
NORMALIZE
THRESHOLD
MAKE BRUSH



MORPHOLOGICAL OPERATOR CLOSING



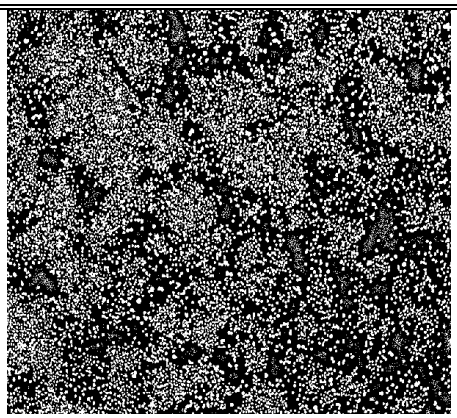
BLUR with MEDIAN

***EXAMPLE 8:*** `hcell –i “./Images” –o “./Output” –f pipeline8.xml`

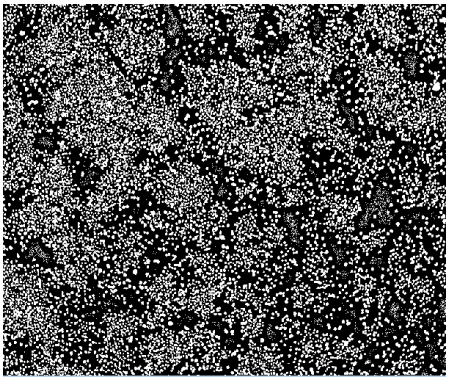
***DESCRIPTION:*** Applies a Laplacian Operator. Threshold now is done with Adaptive thresholding

***SEQUENCE OF FUNCTIONS.***

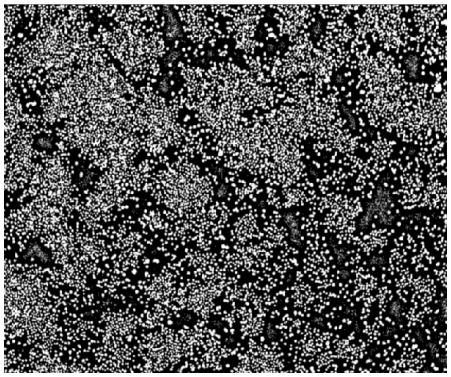
LOAD
CROP
NORMALIZE



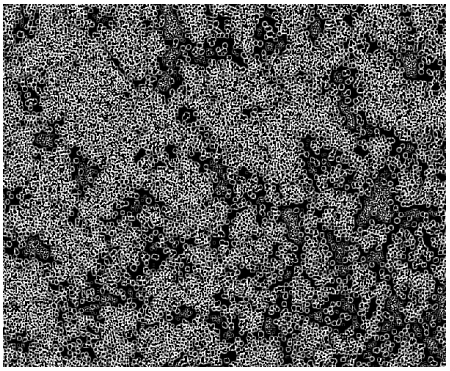
ADAPTIVE THRESHOLD



MAKE BRUSH
ERODE



GBLUR



LAPLACIAN

***EXAMPLE 9:*** `hcell –i “./Images” –o “./Output” –f pipeline9.xml`

***DESCRIPTION:*** Applies a 2D linear filter Operator using a user defined Kernel for sharpening.

***SEQUENCE OF FUNCTIONS.***

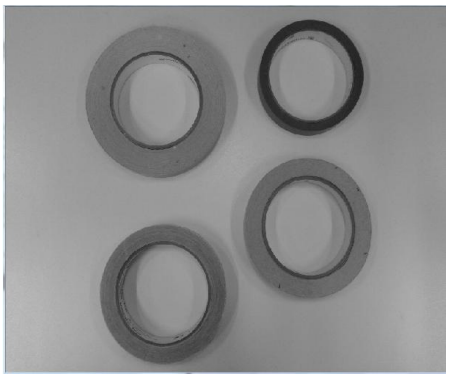
LOAD
CROP
NORMALIZE
THRESHOLD
ERODE

CREATE KERNEL
FILTER 2D (SHARPEN)

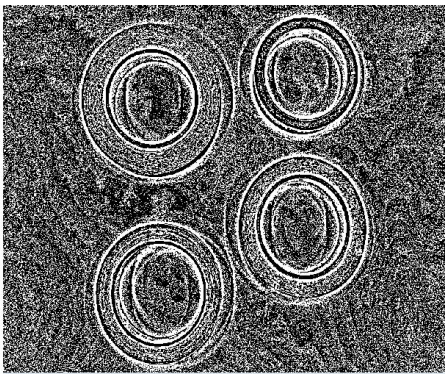
***EXAMPLE 10:*** `hcell –i "./Images" –o "./Output" –f pipeline10.xml`

***DESCRIPTION:*** Applies a Watershed segmentation separating foreground than background.

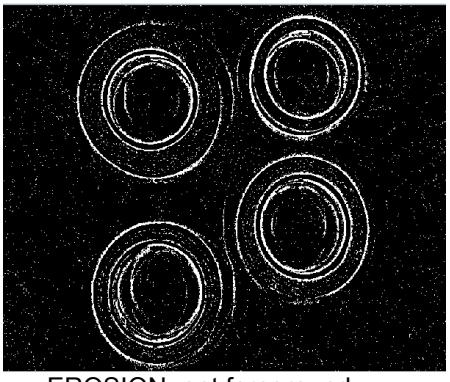
***SEQUENCE OF FUNCTIONS.***



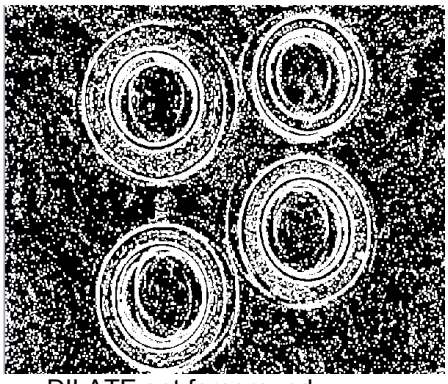
LOAD tape picture.
SHOW tape picture.
NORMALIZE.



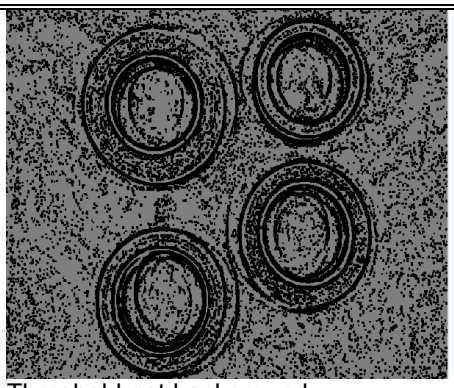
ADAPTIVE THRESHOLD



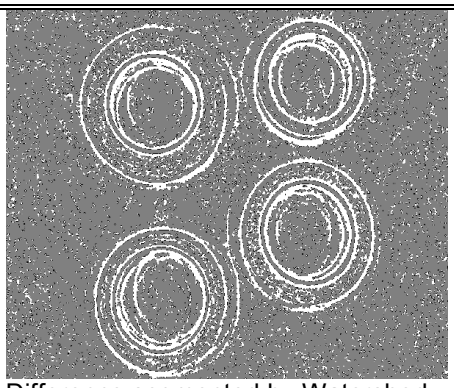
EROSION- get foreground



DILATE get foreground



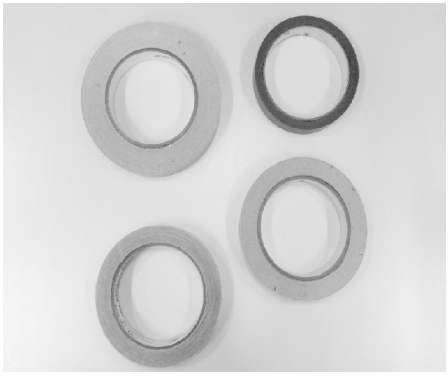
Threshold get background



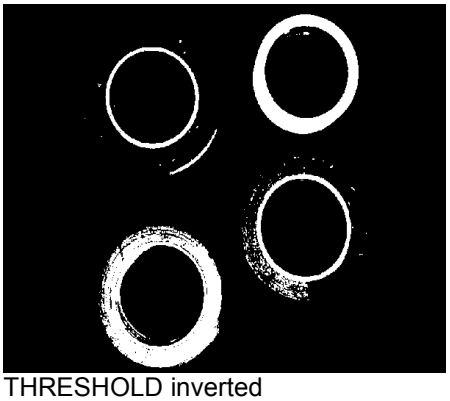
Difference segmented by Watershed

***EXAMPLE 11:*** `hcell -i "./Images" -o "./Output" -f pipeline11.xml`

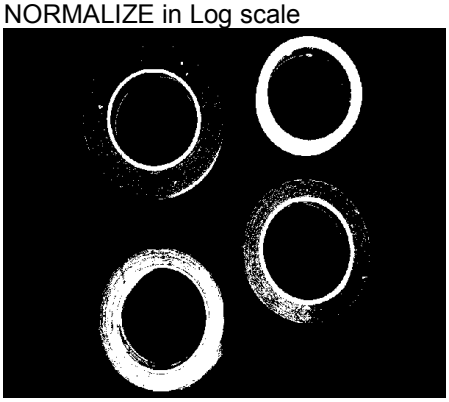
***DESCRIPTION:*** Find contours of the image and then applies filling.
***SEQUENCE OF FUNCTIONS:***



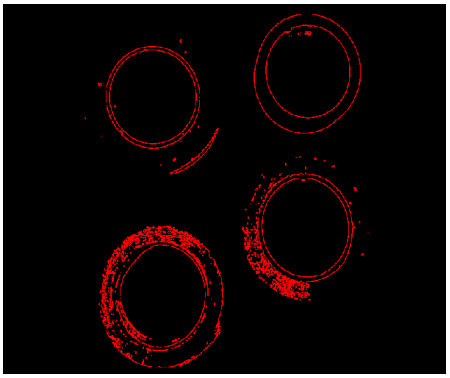
LOAD tape picture.
SHOW tape picture.
NORMALIZE in Log scale



THRESHOLD inverted



OPENING

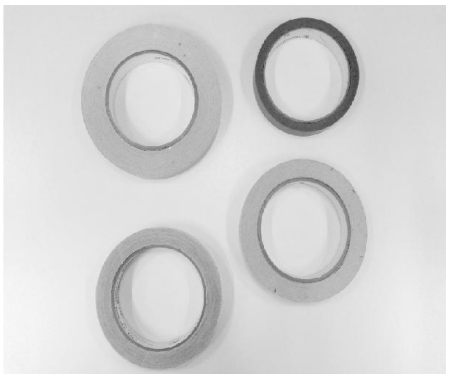


FIND CONTOURS with connected components.

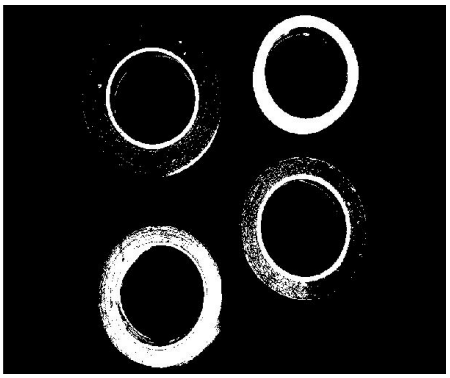
FLOODFILL

***EXAMPLE 12:*** `hcell –i “./Images” –o “./Output” –f pipeline12.xml`

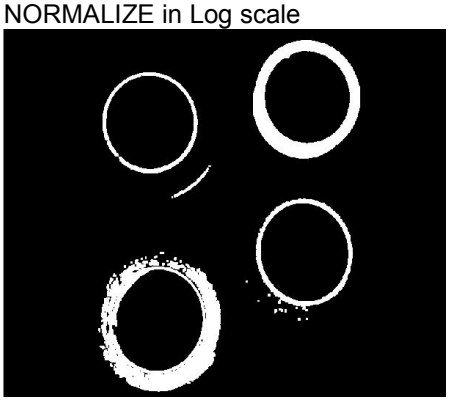
***DESCRIPTION:*** Find contours of the image and then applies filling.
***SEQUENCE OF FUNCTIONS:***



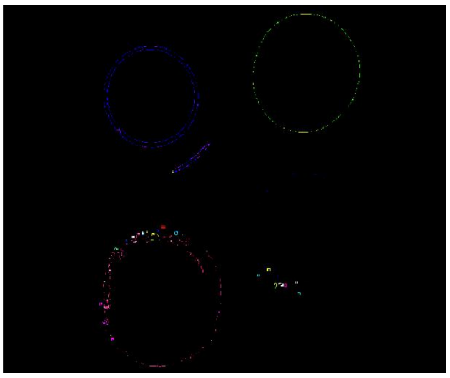
LOAD tape picture.
SHOW tape picture.
NORMALIZE in Log scale



THRESHOLD inverted



OPENING



FIND CONTOURS

BLEND example with opening and contours

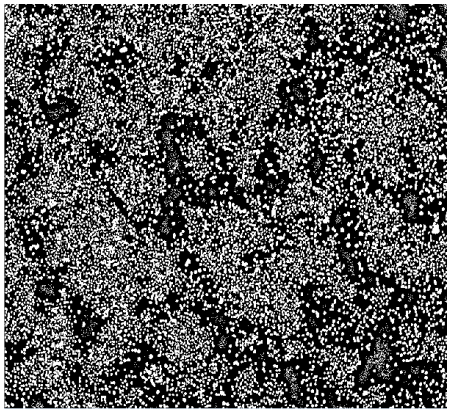
PAINT OBJECTS extracted from contours to the opening image

***EXAMPLE 13:*** `hcell –i “./Images” –o “./Output” –f pipeline13.xml`

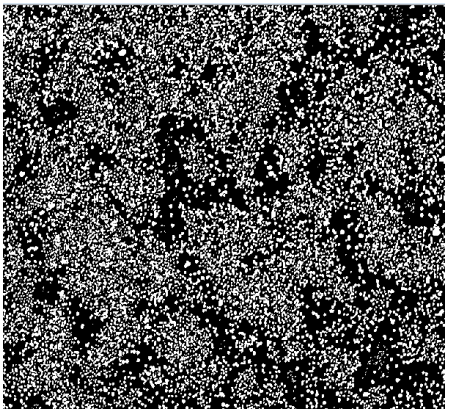
***DESCRIPTION:*** Applies a Laplacian Operator. Threshold now is done with Adaptive thresholding

***SEQUENCE OF FUNCTIONS.***

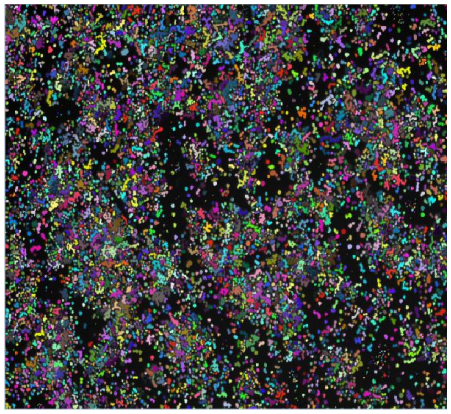
LOAD
CROP
NORMALIZE



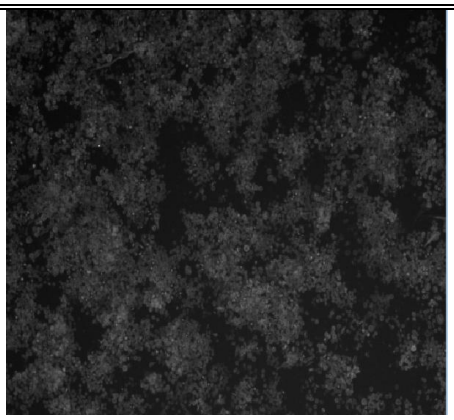
ADAPTIVE THRESHOLD



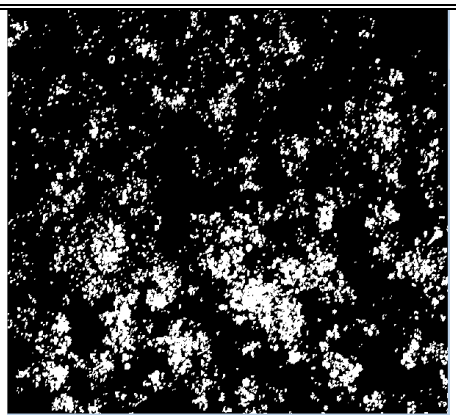
MAKE BRUSH
OPENING



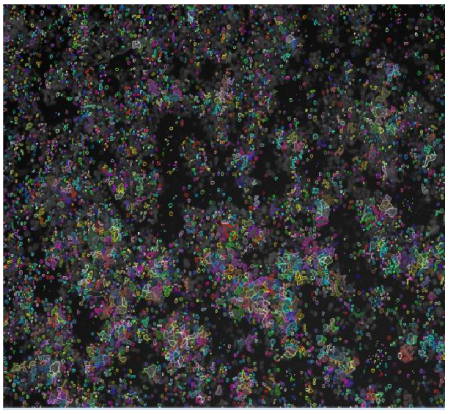
LABEL nuclei
PAINT OBJECTS



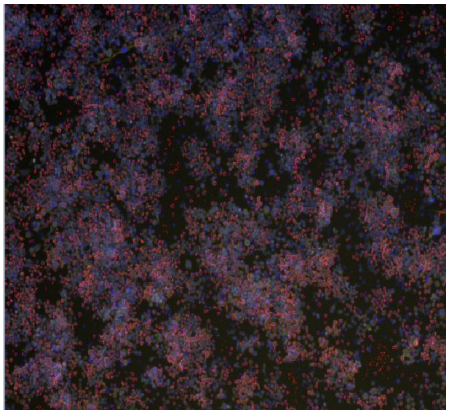
LOAD FITC channel
CROP same size as nuclei
NORMALIZE



THRESHOLD
DILATE



PROPAGATE
PAINT OBJECTS BODIES

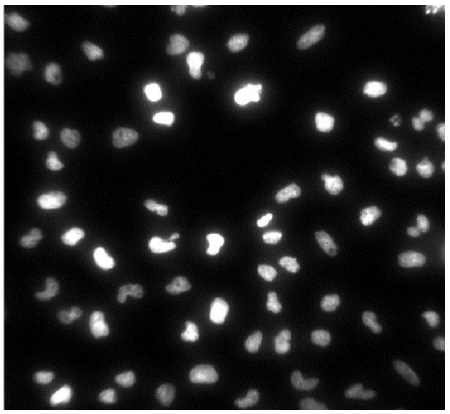


RGB Composite nuclei and body
PAINT OBJECTS from BODIES

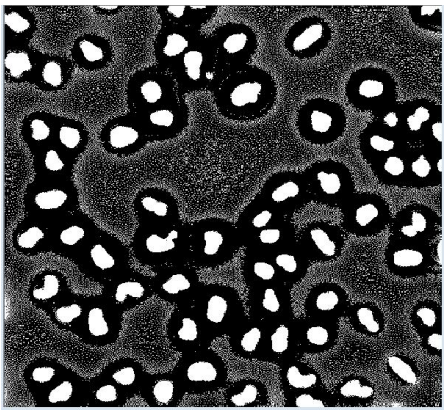
***EXAMPLE 14:*** `hcell –i "./Images" –o "./Output" –f pipeline14.xml`

***DESCRIPTION:*** Takes an image and applies a complete pipeline.

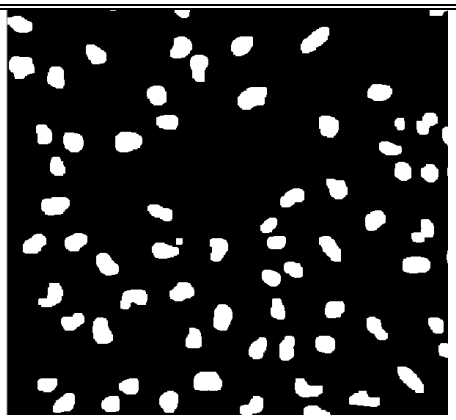
***SEQUENCE OF FUNCTIONS.***



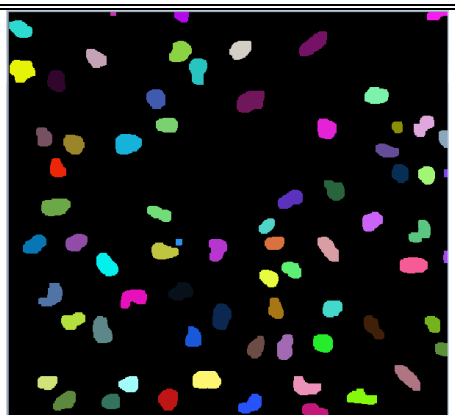
LOAD and NORMALIZE nuclei



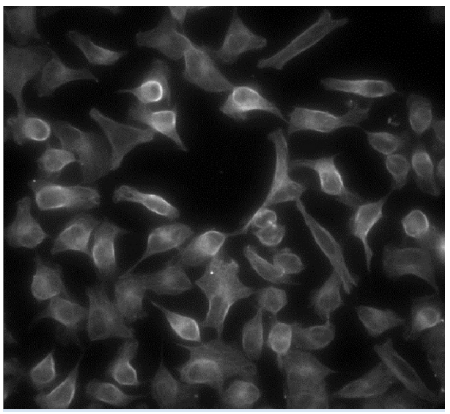
ADAPTIVE THRESHOLD



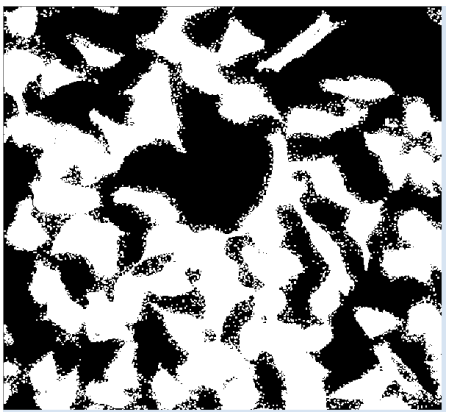
MAKE BRUSH
OPENI NG



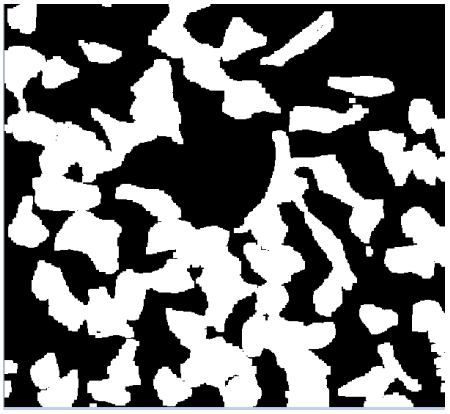
LABEL nuclei
PAINT OBJECTS



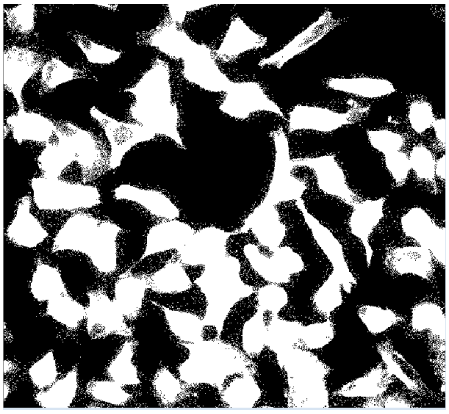
LOAD FITC channel
NORMALIZE



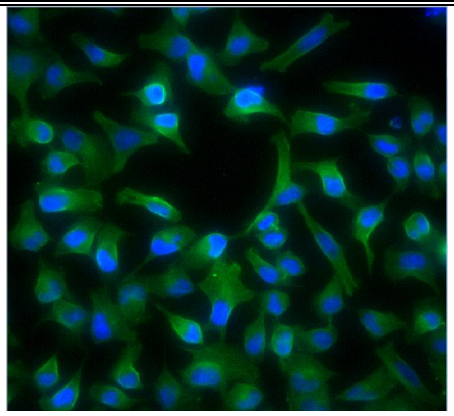
THRESHOLD



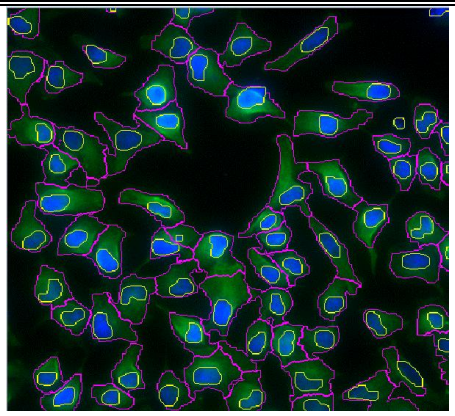
DILATE



OPENING



RGB Composite nuclei and body

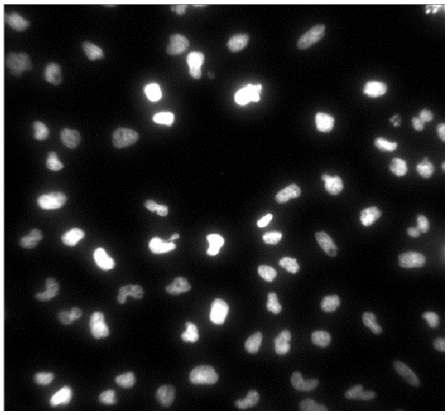


PROPAGATE
PAINT OBJECTS BODIES

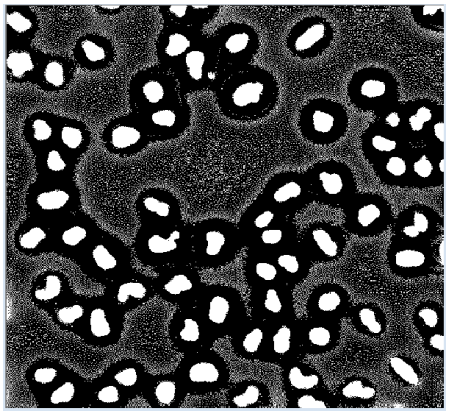
***EXAMPLE 15:*** `hcell –i “./Images” –o “./Output” –f pipeline15.xml`

***DESCRIPTION:*** Extract features from nuclei.

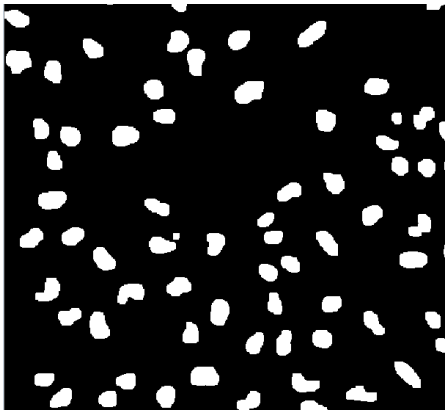
***SEQUENCE OF FUNCTIONS.***



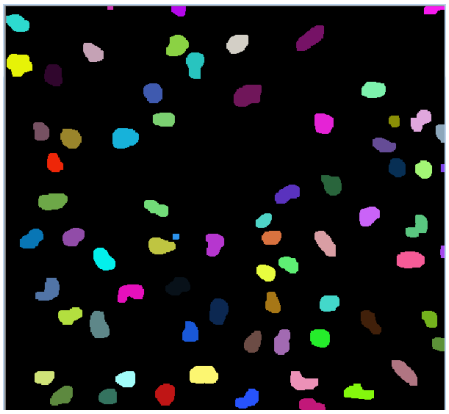
LOAD and NORMALIZE nuclei



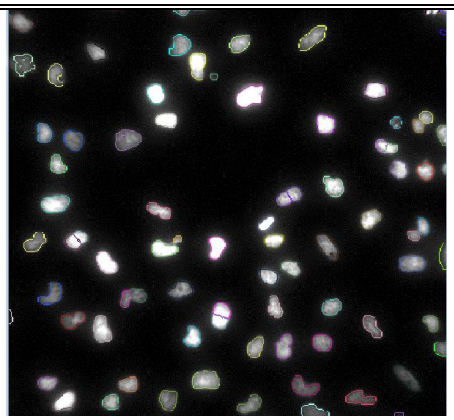
ADAPTIVE THRESHOLD



MAKE BRUSH
OPENI NG



LABEL nuclei
PAINT OBJECTS



SPLIT by WATERSHED

FEATURES BASIC, SHAPE, MOMENT and HARALICK from nuclei objects.

# Chapter 4: For developers: creating a new function.

Once you have download your code and you have set up the project with your favorite compiler, you can start to develop functions adapted to your own needs.

We are happy to add your new functions into our new versions of Hcell. If that is the case, just send us an email with the function XML and the code in a simple text file. We will test the function and we will add it in the next release of the program.

*STEP 1:* Design your XML function following the next format:

```
<MODULE seq="XXX" name="MODULE">
      <FUNCTION name="FUNCTION NAME">
      </FUNCTION>
</MODULE>
```

Now we select our function name. We can have spaces or any character for the function name, there is no problem with that. We also need to select in which module we want to place it. We have 5 available modules:

- File Processing for input and output formatting: load, write, composite images, show...
- Image Processing includes all image processing functions: filters, morphological operators, thresholders…
- Segmentation Processing includes functions that given an image return a set of objects identifying the regions of interest. Functions for filtering or manipulating this objects are also included here.
- Compute Features takes objects from a list (usually coming from Segmentation Processing methods) and extract quantitative values from them, like area, brightness...
- General has functions for the maintenance of the program not related directly to the images.

You will find in the code other 3 more modules: Registration, Analysis and Machine Learning (now empty), which are work in progress, but you are free to add a new function if you think so.

```
<MODULE seq="XXX" name="IMAGE PROCESSING">
      <FUNCTION name="GBLOB">
      </FUNCTION>
</MODULE>
```

Then add your new parameters. Usually it is necessary to specify the input image and the output image. The input image must come from the output of one of the previous functions.We are going to create the GBLOB function, a sort of derivative filter that we will add to the Image processing module.

This is our function as it is programmed in R language.

```
## BLOB. Filter for granulometry. From EBImage package.
## example: blob = gblob(x0=15, n=49, alpha=0.8, beta=1.2)
gblob = function(x0, n, alpha, beta)
 {
  xx = seq(-x0, x0, length.out=n)
  xx = matrix(xx, nrow=length(xx), ncol=length(xx))
  xx = sqrt(xx^2+t(xx)^2)
  z = dnorm(xx, mean=0, sd=alpha) -  0.65*dnorm(xx, mean=0, sd=beta)
  z/sum(z)
 }
```

Now we add our parameters. Besides input and output values, we want to add a block size, alpha, beta and sequence. We can also add the ASHOW for showing our results.

```
<MODULE seq="XXX" name="IMAGE PROCESSING">
      <FUNCTION name="GBLOB">
            <ASHOW>FALSE</ASHOW>
            <WINDOWSNAME>BLOB</WINDOWSNAME>
            <INPUT>IMAGE_X</INPUT>
            <SIZE>49</SIZE>                  ← This is a integer N
            <SEQUENCE>15</SEQUENCE>          ← This is a integer X0
            <ALPHA>0.8</ALPHA>         ← This is a double alpha
            <BETA>1.2</BETA>           ← This is a double beta
            <OUTPUT>BLOB_TRANS_IMAGE_X</OUTPUT>
      </FUNCTION>
</MODULE>
```

You need to be aware of the input types, because if they are different from a string or they are images or objects coming from previous modules, you will need to declare them as integer, double or identifiers (objects existent in the pipeline).

Functions and module names, can have spaces or any Unicode character. Parameter names CANNOT have spaces between them or not alphanumeric characters.

*STEP 2.* Next step is going to the *ImageProcessing.h* file or the .h from the corresponding module. Each new function must be specified in the constructor. The inner name of your function should be similar to the one in the XML, for example, our GBLOB is _gblob (with underscore):

```
ImageProcessing(){

            ///------------------FUNCTIONS------------------------------/////
            tFunc["THRESHOLD"] = &ImageProcessing::_threshold;
            tFunc["ERODE"] = &ImageProcessing::_erode;
            tFunc["DILATE"] = &ImageProcessing::_dilate;
            tFunc["MORPHOLOGICAL OPERATION"] = &ImageProcessing::_morphOp;
            …
            tFunc["MATRIX OPERATION"]=&ImageProcessing::_matOp;
            tFunc["MATRIX TO SCALAR OPERATION"]=&ImageProcessing::_mtsOp;
            // -----Add here your new function---
            tFunc["GBLOB"] = &ImageProcessing::_gblob;
```

*STEP 3*. However, the _gblob function has not been created. We need to do it now. Go to the end of the file and add two headers:

**FIRST HEADER :**

```
void _myfunction(std::vector<MType *> parValues, unsigned int pid)
{
}

In our case:

void _gblob(std::vector<MType *> parValues, unsigned int pid)
{
 }
```

This is what the pipeline is calling first. Parameters from the XML file are inserted using the vector parValues. parValues is a vector of MType. MType is used for emulate reflection (creating data types from strings).

The pid is the *pipeline identifier*. When execution comes, each pipeline has unique identification number. That avoids conflicts between resources. Imagine we have 20 pictures to process, called Image_DAPI, where DAPI is the pattern used for extract them from the directory using regular expressions. We could have something like B1_fld1_DAPI, B2_fld1_DAPI,… B20_fld1_DAPI.

Each workflow, will have a number, between 0 and 19. Then, the program would be 0 B1_fld1_DAPI for workflow 0, 1B2_fld1_DAPI for workflow 1, 2B3_fld1_DAPI for workflow 2, … until 19B20_fld1_DAPI for workflow 20.

Adding the pid to each input and output is necessary, except for cases where data remains constant along all the workflow, like kernels or factors.

That means we need add this pid number in front of the input and output values. If you don´t do that, objects in the pool will not be recognized properly and the execution will finish with an error. Needs to be passed as parameter when and identifier type is used. An identifier is an input or output image or object.

We can extract our data like this:

```
void _gblob(std::vector<MType *> parValues, unsigned int pid)
{
	double alpha  = (dynamic_cast<MDoubleType*>(parValues[0]))->getValue(pid);
	bool ashow    = (dynamic_cast<MBoolType*>(parValues[1]))->getValue();
	double beta   = (dynamic_cast<MDoubleType*>(parValues[2]))->getValue(pid);
	string input  =
		  (dynamic_cast<MIdentifierType*>(parValues[3]))->getValue(pid);
	MIdentifierType* out = dynamic_cast<MIdentifierType*>(parValues[4]);
	string output = out->getValue(pid);
	int sequence = (dynamic_cast<MIntType*>(parValues[5]))->getValue(pid);
	int size = (dynamic_cast<MIntType*>(parValues[6]))->getValue(pid);
	const char* wName = (dynamic_cast<MStringType*>(parValues[7]))->getValue();
```

Be aware of three facts here:

- Input values are **sorted in alphabetic order.** That means, that alpha will get the first value, then ashow, then beta and so on. Be aware of the vector index. The order in the XML file is not relevant, values are sorted alphabetically after being read.
- Data types must be concordant. If my input is a double, I will cast my input value to a double.
- Strings and const char* can be used indistinctly, but if a function uses a const char*, then use the function string.c_str() to transform the string to a constant char array.
- Double,int and Identifier type need the pid as parameter.

*STEP 4*. We just need one thing more, the call to our second header. The second header is a function that really operates with the values.

```
__gblob(input.c_str(),output.c_str(), size, sequence, alpha, beta);
```

This is the final code:

```
/*********************************

GBLOB granulometry transform.

Applies a derivative transform filter to the image.

***************/
void _gblob(std::vector<MType *> parValues, unsigned int pid)
{
	double alpha  = (dynamic_cast<MDoubleType*>(parValues[0]))->getValue(pid);
	bool ashow    = (dynamic_cast<MBoolType*>(parValues[1]))->getValue();
	double beta   = (dynamic_cast<MDoubleType*>(parValues[2]))->getValue(pid);
	string input  =
		  (dynamic_cast<MIdentifierType*>(parValues[3]))->getValue(pid);
	MIdentifierType* out = dynamic_cast<MIdentifierType*>(parValues[4]);
	string output = out->getValue(pid);
	int sequence = (dynamic_cast<MIntType*>(parValues[5]))->getValue(pid);
	int size = (dynamic_cast<MIntType*>(parValues[6]))->getValue(pid);
	const char* wName = (dynamic_cast<MStringType*>(parValues[7]))->getValue();

	  __gblob(input.c_str(),output.c_str(), size, sequence, alpha, beta);

	if(ashow)
	  {
		this->display(output,wName);
	  }

	out->refresh(pid);

}
```

The last step uses the ashow Boolean value: if users wants to display, the display function is called and a window with the resultant image will be shown.

If we have an output image, is possible that we want to operate with it. For example, I want to store the inverted result of my image or apply a mask (using and OR operator). The function refresh updates the value obtained in the original function an applies the corresponding operations.

## *STEP 5*.

Before finishing this part, we need to do something else. We need to tell Hcell if parameters have different values from strings. Go to the file *Action.cpp* . In the function Initialize, you will find a map where types are assigned to each value. Let´s do it:

```
/**GBLOB options**/
	amap["SEQUENCE"]=intT;
	amap["BETA"]=doubleT;
```

We don´t need to create the ASHOW, WINDOWS_NAME, INPUT, OUTPUT, ALPHA and SIZE values because they are already mapped with the right types. Be aware that you are creating reserved words, so if you create new functions, you can use the same word, but the type is fixed. You can´t use SIZE as a double, you will need to rename it using a new reserved word called, for example, DSIZE.

The following types can be assigned: **stringT (default), idT, intT, doubleT, boolT.**

## *STEP 6*.

**SECOND HEADER:**

```
void __gblob(const char* input,const char* output, int size,int sequence, double
alpha, double beta)
{
}
```

The real function. Do the same as you will program it in C++. Only difference is that input and output must passed as a const char pointers or strings (use one or the other makes no difference).

Notice that the first header is underscore + name of the function. Second header is double underscore + name of the function. The reason of this is simple. I could go now and call an external function called gblob, that could be an external function.

```
void __gblob(const char* input,const char* output, int size,int sequence, double
alpha, double beta)
{
-- TO DO before start.
vector<vector<double>> my_newdata = gblob(sequence, size, alpha, beta);
-- TO DO at the end.

}
```

Now, let´s go to complete the previous functions:

**TO DO before start.**

- Add a trace function

```
ut::Trace tr = ut::Trace("GBLOB",__FILE__);
```

This function creates a trace object. It is useful to print things by console, that only will be printed in verbose mode.

- Load resources from pool.

  ```
  Mat *src;
  src = pool->getImage(input);
  ```

  Functions from pool are
  - getImage for Mat images (from OpenCV),
  - getlObj for vloPs, a matrix, where each row is an object. An object is called a loP, list of Points. A vloP is a vector of list of Points.
  - getFactor. for simple scalar values, stored as doubles.

In our case we need to get the input image.

**TO DO at the end.**

- Add resources to the pool.

  ```
  pool->storeImage(final_image,output);
  return;
  ```

  Our resultant image or vector of objects must be stored again in the pool for later usage.

```
void __gblob(const char* input,const char* output, int size,int sequence, double
alpha, double beta)
{
	ut::Trace tr = ut::Trace("GBLOB",__FILE__);

	Mat *src;
	src = pool->getImage(input);

	// sequence from -15 to 15, 49 times
	vector<double> xx;

	double by = (2.0*sequence)/((float)size-1.0);
	float init = (float) -sequence;
	while(init<=sequence)
	{
	  xx.push_back(init);
	  init+=by;
	}
	// Create our matrix
	 Mat kernel = Mat::zeros( size, size, CV_64F );
	// and initialize it
	 MatIterator_<double> it;
	 vector<double>::iterator itxx;


	double aux;
	for( size_t i = 0; i < size; i++ )
	{
		aux = xx[i];
```

```
          for( size_t j = 0; j < size; j++ )
          {
                      kernel.at<double>(i,j)=aux;
          }
        }

        Mat tker;
        transpose(kernel,tker);
        cv::pow(kernel,2,kernel);
        cv::pow(tker,2,tker);
        cv::add(kernel,tker,kernel);
        cv::sqrt(kernel,kernel);

        double x;
        double sum=0.0;

        for( size_t i = 0; i < size; i++ )
        {
          for( size_t j = 0; j < size; j++ )
           {
               x = kernel.at<double>(i,j);
               kernel.at<double>(i,j)=normal_pdf(x,0,alpha)-
0.65*normal_pdf(x,0,beta);
               sum+= kernel.at<double>(i,j);
          }
        }
        kernel /=sum;
        Point anchor(kernel.cols - kernel.cols/2 - 1, kernel.rows - kernel.rows/2 -
1);
    int borderMode = BORDER_CONSTANT;
        flip(kernel,kernel,0);
        Mat final_image;
        filter2D(*src, final_image, -1,kernel, anchor, 0, borderMode);
        final_image/=2;

        pool->storeImage(final_image,output);
        return;
}
```

Notes to implementation:

- OpenCV is used. If you want to program for Hcell, you need to consult OpenCV help. In most cases using functions is straightforward, but you need to be careful with data types between matrix conversions. In our case we use a 64F double precission. Be aware that common types are 8U, 16U and 32F, and this is a special case. Only use 64F if you really need it, and then, in next functions, don´t forget to convert your matrix to the adequate input data type.
- `normal_pdf` is a function that returns the probabilistic distribution function of a normal distribution. Any new function that you think could be useful for later, you can store it as a static function in the proctools class (proctools.h).

## *STEP 8*.

Only the first part of this step is mandatory: register to the function to the checker. The Checker class takes the list of parameters and checks them before the pipeline runs to make sure that values are correct.
Go to the file *Checker.cpp* and add, to the corresponding module (`Checker::ParamsIP`

in our case, for Image Processing), the function parameters at the end of the matrix.

```
        {"GBLOB","INPUT","ASHOW","SIZE","WINDOWSNAME","SEQUENCE","ALPHA","BETA","OU
TPUT","-"},
```

It is necessary to increase in *Checker.h* the number of functions.

```
#define TOTAL_FUNCTIONS_IP 20 -> #define TOTAL_FUNCTIONS_IP 21
```

Next steps are necessary for debugging. The process is quite similar to what we already have done. Go to *Checker.h* and first subscribe the function in the constructor *Checker()* and create and empty function called *_checkYourFunctionName* :

```
cFunc["GBLOB"] =  &Checker::_checkGBlob;
```

The function must have the next format: function which takes as input the action (where the parameters are stored) and the error by reference.

```
bool _checkGBlob(Action *action, string &error){
```

Then we retrieve the parameters from the action (similar to what we did before) and we subscribe the function to a Map. This map keeps track of the inputs and outputs for each function.

```
        vector<MType *> params = action->getParameters();
        long id=this->myMap.subscribeFunction(action->getCurrentAction());
```

Now we can check the inputs and the outputs:

```
        if(!myMap.addImageInput(id,params[3]->getValue(),error)) return false;
        if(!myMap.addImageOutput(id,params[4]->getValue(),error)) return false;

                return true;
        };
```

The number of the parameter is the same we got when we create the caller function in the Step 3.

Now we can check in our function what we want to check. For example, Sequence must be a positive integer and Size must be a kernel. We can use the following functions:

```
bool checkRegExp(const char* regexp,string &error)
```
Checks for a valid a regular expression from a string.

```
bool checkType(std::map<string,int> parmap,string value,string &error)
```
Checks for a right type. The type must exist in the eMap map from the corresponding module.

```
bool checkRangeifType(std::map<string,int> parmap,string type,int expected_type,
double param,double max, double min,string &error,bool strict=false)
```

For an specific type given in expected_type, if this type is selected, then checks in the variable param if the range between max and min is correct. The Boolean strict = true implies the range is ]min, max] (For example, from 0 to 5 without considering 0). By default strict = false, which implies ]min,max[ .

```
bool checkRange(double param,double max, double min,string &error)
```

Same as before, but independent of the selected type. Valid for doubles and integers.

```
bool checkColour(string colour,string &error)
```

Checks if the string contains a valid format for a colour.

```
bool checkMatrix(map<string,int> param,string matrix,int kernelsize,string &error)
```

Checks if the string contains a valid format for a matrix. First checks if a reserved word of a type is used instead (like Zeros, Eye or Ones).

```
bool checkKernel(int ksize,string &error)
```

Checks for a valid size of a kernel (odd values and in a proper integer range that doesn´t exceed the image size).

```
bool checkEmpty(string param,string &error)
```

Checks that the string is empty or not. Sometimes needed, but is done internally by the XMLChecker,too.

In our case we will use *checkRange* and *checkKernel.*

```
if(!checkRange((dynamic_cast<MIntType*>(params[5]))->getValue(),1000,1,error=
"Parameter SEQUENCE. \n")) return false;

if(!checkKernel((dynamic_cast<MIntType*>(params[6]))->getValue(),error="Parameter
KERNEL SIZE. \n")) return false;
```

It is convenient to write information about the parameter in the error string.

## *STEP 9*.

FINAL: Testing our function.
Now we can create our pipeline and run it!

```
<PIPELINE>
	<MODULE seq="001" name="FILE PROCESSING">
	<FUNCTION name="LOAD FILE">
			<REGEXP>(.*)Cy3(.*)</REGEXP>
			<OUTPUT>IMAGE1_CY3</OUTPUT>
	</FUNCTION>
	</MODULE>
	<MODULE seq="002" name="FILE PROCESSING">
		<FUNCTION name="NORMALIZE">
			<INPUT>IMAGE1_CY3</INPUT>
			<OUTPUT>IMAGE2_CY3</OUTPUT>
			<ASHOW>TRUE</ASHOW>
			<WINDOWSNAME>NORM_CY3</WINDOWSNAME>
			<TYPE>MAXMIN</TYPE>
			<MAXINT>0.03</MAXINT>
			<MININT>0.002</MININT>
		</FUNCTION>
	</MODULE>
	<MODULE seq="003" name="IMAGE PROCESSING">
		<FUNCTION name="GBLOB">
			<ASHOW>TRUE</ASHOW>
			<WINDOWSNAME>BLOB</WINDOWSNAME>
```

```
                    <INPUT>IMAGE2_CY3</INPUT>
                    <SIZE>49</SIZE>
                    <SEQUENCE>15</SEQUENCE>
                    <ALPHA>0.8</ALPHA>
                    <BETA>1.2</BETA>
                    <OUTPUT>BLOB_TRANS_IMAGE_X</OUTPUT>
                </FUNCTION>
            </MODULE>
</PIPELINE>
```

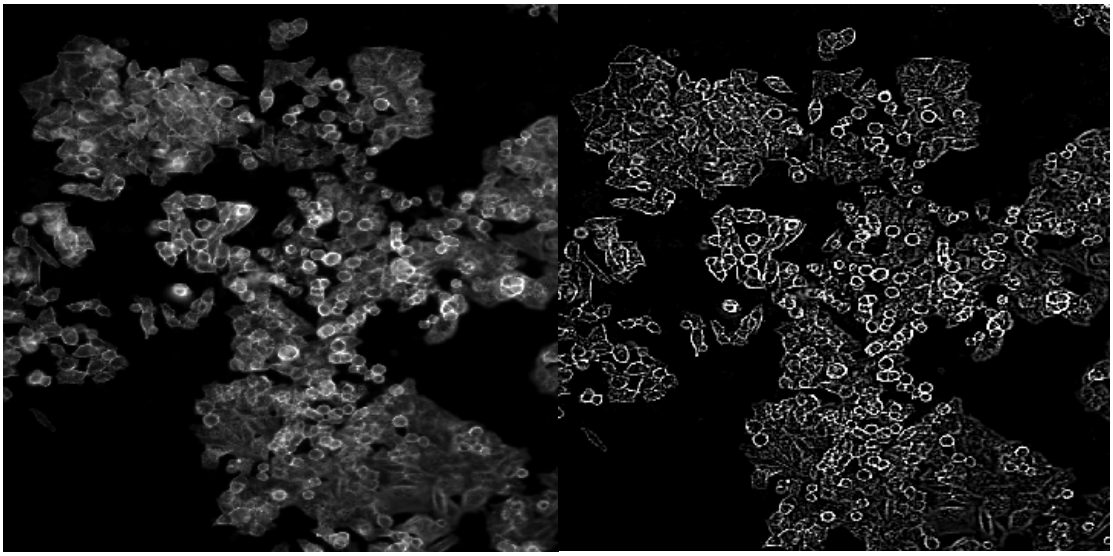


*Image before and after applying our GBLOB function.*

# Chapter 5: Functions Description

| Version | Date | Author | Notes |
|---|---|---|---|
| 0.7 | 03.2014 | José Miguel Serra Lleti | |

*Some of the information presented in this manual relative to functions description has been extracted from the pages of OpenCV docs.opencv.org, with © Copyright belonging to opencv dev team.*

## 1.1 File Processing Module

### 1.1.1 LOAD FILE

Loads files from the input directory based on a regular expression. All files matching that regular expression are loaded for processing.

NOTE: it doesn´t search on subdirectories.

| LOAD FILE | Description | Default assignment |
|---|---|---|
| REGEXP | String – regular expression, will search over the directory given as input directory and will search for them.<br>Be aware regular expressions are following the rules denoted in:<br>http://www.cplusplus.com/reference/regex/ECMAScript/ | . (current directory any file) |
| OUTPUT | String naming a resultant image from load. | “” |

Example:

```
<MODULE seq="001" name="FILE PROCESSING">
            <FUNCTION name="LOAD FILE">
                  <REGEXP>^[D-M](.*)- 2(.*)DAPI(.*) </REGEXP>
                  <OUTPUT>IMAGE1</OUTPUT>
            </FUNCTION>
</MODULE>
```

Use PIPELINE1.xml as example.

Further versions will include 8 or 16 bits depth and channel selection (gray or rgb), but by default selection is made automatically. Be aware: if you load a color image, some functions only can process gray scale images. Read function documentation before use it. If something is wrong use verbose mode and check the depth and number of channels of your loaded image.

### 1.1.2 WRITE FILE

WRITE FILE saves a generated image into a specified file. The image format is chosen based on the filename extension specified in the FILEEXT field.

| WRITE FILE | Description | Default assignment |
|---|---|---|
| INPUT | String | |
| FILEEXT | String 3 letters. | Possible values:<br>PNG, JPG, JPEG, PBM, PGM, PPM,TIF,TIFF<br>Depending of saved file type. |

| | | |
|---|---|---|
| P_COMPRESSIONLEVEL | In JPG - Integer range between 0 and 100 (default 95). Higher the better quality.<br>In PNG - between 0 and 9 for PNG. 9 means more compressed (less size). Default is 3. | Only can be used with PNG and JPG. |
| FOUTPUT | String | Filename. |
| APPEND_ORIGINAL_NAME | Boolean.<br>Adds original name from file. If only one channel exists (e.g. DAPI), the whole file name is added. If several channels are used, the common substring between both will be used.<br>E.g.<br>A-2-fld1 DAPI<br>A-2-fld1 Cy3<br>A-2-fld1 FITC<br>Then, A-2-fld1 is added to the name file. | FALSE |
| APPEND_DATE | Boolean.<br>Appends date to the end of the file with the format :<br>YYYY – MM – DD | FALSE |

By default a counter is added for differentiate pictures, which are always processed in alphanumeric order.

```
<MODULE seq="0xx" name="FILE PROCESSING">
      <FUNCTION name="WRITE FILE">
            <INPUT>IMAGE_COMPOSITE</INPUT>
            <FILEEXT>PNG</FILEEXT>
            <P_COMPRESSIONLEVEL>6</P_COMPRESSIONLEVEL>
            <FOUTPUT>IMAGE_COMPOSITE</FOUTPUT>
            <APPEND_ORIGINAL_NAME>TRUE</APPEND_ORIGINAL_NAME>
            <APPEND_DATE>TRUE</APPEND_DATE>
      </FUNCTION>
</MODULE>
```

Example of output : IMAGE_COMPOSITE_H - 4(fld 1 wv DAPI - DAPI)_4_2013-10-15.tiff

### 1.1.3 NORMALIZE

Normalize an image, i.e., rescales the values between 0 and 1 (or a minimum and a maximum).

| NORMALIZE | Possible Values | Default assignment |
|---|---|---|

| | | |
|---|---|---|
| INPUT | Name of source image. | |
| OUTPUT | Name of destiny image. | |
| ASHOW | Shows the result of applying normalization to the image. | FALSE |
| WINDOWSNAME | Window's name of shown image. | |
| TYPE | MAX<br>MAXMIN<br>LOG<br>AUTO<br>CLAHE | |
| MAXINT | Maximum intensity value considered in MAXMIN. Values bigger than MAXINT are changed to MAXINT.<br>In CLAHE maxint is the number of clip limits. | 1 |
| MININT | Minimum intensity value considered in MININT. Values smaller than MININT are changed to MININT. | 0 |

Usage example:

```
<MODULE seq="002a" name="FILE PROCESSING">
      <FUNCTION name="NORMALIZE">
            <INPUT>IMAGE1_DAPI</INPUT>
            <OUTPUT>IMAGE2_DAPI</OUTPUT>
            <ASHOW>FALSE</ASHOW>
            <WINDOWSNAME>NORM_DAPI</WINDOWSNAME>
            <!-- MAXMIN or LOG-->
            <TYPE>MAX</TYPE>
            <MAXINT>1</MAXINT>
            <MININT>0</MININT>
      </FUNCTION>
</MODULE>
```

If the type chosen is MAX:

- The output is a 32 float image between 0 and 1, and all values are simply divided by the maximum value. If you want a

If the type chosen in MAXMIN:

- Normalization is affecting both INPUT and OUTPUT. INPUT image is transformed into the same range than the original one but normalized; OUTPUT is always a 32 float image between 0 and 1. For example, if IMAGE1_DAPI (input) is a 16 bit image, now, IMAGE1_DAPI has normalized values between 0 the minimum and 65535 the maximum and IMAGE2_DAPI (output), now has values between 0 minimum and 1 maximum. The original image is not preserved.
- MAXMIN can use the values MAXINT and MININT to limit the range of intensity in a scale between 0 and 1.
- MAXMIN applies for each pixel the next formula:

  MAXMIN = *(value – min)/ (max-min)*

If the type chosen is LOG:

- Normalization uses a logarithmic function.
  LOG = *1- log (value)/log(max/min)*
- Same considerations as MAXMIN, INPUT image is transformed into the same range than the original one but normalized; OUTPUT is always a 32 float image between 0 and 1. Original image is not preserved.

If the type chosen is AUTO:

- It uses a similar algorithm that is used in FIJI (ImageJ) when we apply the Auto Brightness and Contrast button. More information can be found at:

http://cmci.embl.de/documents/120206pyip_cooking/python_imagej_cookbook#automatic_brightnesscontrast_button

If the type chosen is CLAHE:

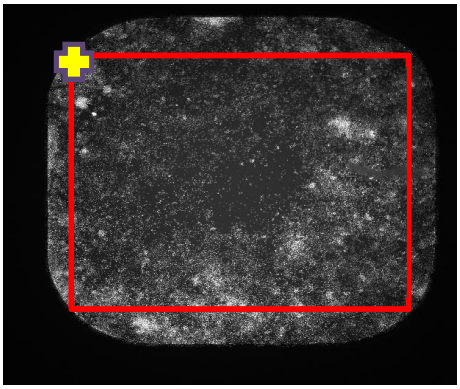
Contrast Limited Adaptive Histogram Equalization. Image is divided into small blocks called "tiles" (tile size is 8x8 by default). Then each of these blocks are histogram equalized. So in a small area, histogram would confine to a small region (unless there is noise). If noise is there, it will be amplified. To avoid this, **contrast limiting** is applied. If any histogram bin is above the specified contrast limit (by default 40), those pixels are clipped and distributed uniformly to other bins before applying histogram equalization. After equalization, to remove artifacts in tile borders, bilinear interpolation is applied.

The clip limit can be adjusted using maxint and the tile size using the minint values instead.

### 1.1.4 CROP IMAGE

Crops a ROI in the image, using a box.

| CROP IMAGE | Possible Values | Default assignment |
|---|---|---|
| INPUT | Name of source image. | |
| WINDOW_WIDTH | Width of the ROI, crop in the x direction. | 0 |
| WINDOW_HEIGHT | Height of the ROI, crop in the y direction. | 0 |
| COORDINATES_X<br>COORDINATE_Y | The crop is a box, which reference point is a point in the left upper corner. These are the coordinates of that origin point. (Yellow cross in the image below.) | 0,0 |
| ASHOW | Shows the result of applying normalization to the image. | FALSE |
| WINDOWSNAME | Name of the generated window by the ASHOW command. | "" |
| OUTPUT | Name of destiny image. | |



```
<!-- BORDER REMOVAL: Cropped to ideal size to remove borders -->
    <MODULE seq="001b" name="FILE PROCESSING">
            <FUNCTION name="CROP IMAGE">
                  <INPUT>NUCLEI_ORIGINAL</INPUT>
                  <WINDOW_WIDTH>1500</WINDOW_WIDTH>
                  <WINDOW_HEIGHT>1400</WINDOW_HEIGHT>
                  <COORDINATES_X>330</COORDINATES_X>
                  <COORDINATES_Y>330</COORDINATES_Y>
                  <ASHOW>FALSE</ASHOW>
                  <WINDOWSNAME>YOU CROPPED ME!</WINDOWSNAME>
                  <OUTPUT>NUCLEI_CROPPED</OUTPUT>
            </FUNCTION>
      </MODULE>
```

### 1.1.5 RGB COMPOSITE

Given three level gray value images, mixes them into one image. Each gray image is placed on each channel and displayed in such a color.

| RGB COMPOSITE | Possible Values | Default assignment |
|---|---|---|
| ASHOW | Shows the result of applying normalization to the image. | FALSE |
| RED_CHANNEL | Name of input image for red channel. | “NULL” |
| GREEN_CHANNEL | Name of input image for green channel. | “NULL” |
| BLUE_CHANNEL | Name of input image for blue channel. | “NULL” |
| WINDOWSNAME | Name of window where result will be shown. | “” |
| OUTPUT | Name of destiny image. | “” |

Example:

```
      </MODULE>
            <MODULE seq="006P" name="FILE PROCESSING">
            <FUNCTION name="RGB COMPOSITE">
                  <RED_CHANNEL>IMAGEN_CY3</RED_CHANNEL>
```

```
                <GREEN_CHANNEL>IMAGEN_FITC</GREEN_CHANNEL>
                <BLUE_CHANNEL>IMAGEN_DAPI</BLUE_CHANNEL>
                <ASHOW>FALSE</ASHOW>
                <WINDOWSNAME>IMAGE COMPOSITION</WINDOWSNAME>
                <OUTPUT>IMAGE_COMPOSITE</OUTPUT>
        </FUNCTION>
</MODULE>
```

Careful: ONLY GREY LEVEL VALUE IMAGES are allowed as input.

In case we want one of our channels empty, we need to fill it with the word NULL. Channel for that then will be empty. Example:

```
                <RED_CHANNEL>IMAGEN_CY3</RED_CHANNEL>
                <GREEN_CHANNEL>NULL</GREEN_CHANNEL>
                <BLUE_CHANNEL>NULL</BLUE_CHANNEL>
```

Only Red channel is shown.

Output of RGB image can be used as input canvas for PAINT OBJECTS function.

### 1.1.6 **BLEND**

Blending is based in the addWeighted function of OpenCV. For more information check: http://docs.opencv.org/doc/tutorials/core/adding_images/adding_images.html

Basically we are using a pixel operator, the linear blend operator. This formula is applied:

$$g(x) = (1 - \alpha)f_0(x) + \alpha f_1(x)$$

By varying $\alpha$ from $0 \rightarrow 1$ this operator can be used to perform a temporal cross-dissolve between two images.

| BLEND | Possible Values | Default assignment |
|---|---|---|
| ASHOW | Shows output image result. | FALSE |
| WINDOWSNAME | Name of output image. | “” |
| INPUT_1 | First input image. | “” |
| INPUT_2 | Second input image. | “” |
| ALPHA_1 | Weight used on the first image. Is equivalent to the 1-alpha in the equation. | “0” |
| ALPHA_2 | Weight used on the second image. Equivalent to the alpha in the equation. | “1.0” |

| | | |
|---|---|---|
| OUTPUT | Output image with same size and number of channels as the input image. | |

NOTE: It is recommended that ALPHA_1 and ALPHA_2 would be complementary, like 0.2, 0.8 or 0.3,0.7, following the blending equation shown before.

```
</MODULE>
        <MODULE seq="006" name="FILE PROCESSING">
        <FUNCTION name="BLEND">
                <ASHOW>TRUE</ASHOW>
                <WINDOWSNAME>BLEND</WINDOWSNAME>
                <INPUT_1>IMAGE2</INPUT_1>
                <INPUT_2>OBJECTS_IM</INPUT_2>
                <ALPHA_1>0.5</ALPHA_1>
                <ALPHA_2>0.5</ALPHA_2>
                <OUTPUT>IMAGEX</OUTPUT>
        </FUNCTION>
</MODULE>
```

### 1.1.7 PAINT OBJECTS

Given a set of objects,

| PAINT OBJECTS | Possible Values | Default assignment |
|---|---|---|
| ASHOW | Shows the result of applying normalization to the image. | FALSE |
| INPUT | Input image used as canvas for painting objects. | "" |
| COLOUR | RED, GREEN, BLUE,<br>GREY, BROWN, PURPLE,<br>BLACK, WHITE<br>YELLOW, MAGENTA, CYAN,<br><br>Any combination from 0-255 in the format:<br>RGB(255,255,255)<br>RGB(0,128,20)<br><br>Any hexadecimal number coding a color. Ex:<br>#FFFFFF<br><br>You check for more colors here:<br>http://www.javascripter.net/faq/rgbtohex.htm | WHITE |

| | | |
|---|---|---|
| THICKNESS | Default value is 0.<br>When thickness is bigger than 0, then, contour mode is on and objects are surrounded by a thick lines. Thickness of this lines in pixels can be controlled with integer values:<br>1,2,3,4,….<br><br>WARNING: Use always 0 when you want to use PAINT_CONTOURS (TRUE). | 0 |
| LOAD_OBJECTS | Vector of objects as input. | “” |
| PAINT_CONTOURS | If TRUE, contours are painted. Contour finding is applied to each object using 8 connected pixel-components. If FALSE objects are filled. | “” |
| OUTPUT | Output image where the output can be stored. Several Outputs can be concatenated in order to create composite images with different types of highlighted objects. | “” |
| WIDTH | Width of windows name when show is true. If not, it specifies the width of the image where objects are going to be painted. | 800 |
| HEIGHT | Height of windows name when show is true. If not, it specifies the height of the image where objects are going to be painted. | 600 |
| WINDOWSNAME | Windows name when output image is shown. | “” |

Conditions:

If you don’t want unexpected behavior, we recommend use this configurations set.

- **COLOUR** – RANDOM, **PAINT CONTOURS** – TRUE, **THICKNESS** – 0
  If thickness is bigger than 0, you get an error message. This problem would be solved in further versions of HCell.

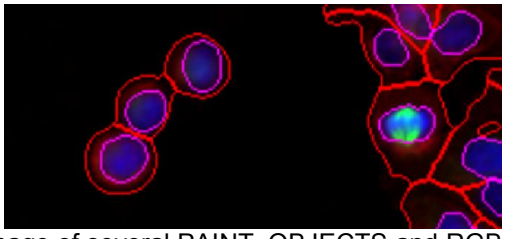
- **COLOUR** – ANY (except RANDOM), **PAINT CONTOURS** – TRUE, **THICKNESS** – 0
  If thickness is bigger than 0, you will probably get the whole objects completely painted.

- **COLOUR** – ANY, **PAINT CONTOURS** – FALSE, **THICKNESS** – 0
- **COLOUR** – ANY, **PAINT CONTOURS** – FALSE, **THICKNESS** > 0
  If thickness is bigger than 0, you will paint the thickness borders.

```
<MODULE seq="008d" name="FILE PROCESSING">
            <FUNCTION name="PAINT OBJECTS">
                  <ASHOW>TRUE</ASHOW>
```

```
                <WINDOWSNAME>Painted nuclei</WINDOWSNAME>
                <INPUT>IMAGE2_DAPI</INPUT>
                <COLOUR>RANDOM</COLOUR>
                <THICKNESS>0</THICKNESS>
                <LOAD_OBJECTS>NUC_OBJ_FINAL</LOAD_OBJECTS>
                <PAINT_CONTOURS>FALSE</PAINT_CONTOURS>
                <OUTPUT>IMAGE_PAINTED_FINAL</OUTPUT>
                <WIDTH>800</WIDTH>
                <HEIGHT>600</HEIGHT>
        </FUNCTION>
</MODULE>
```

Example:



COMPOSITE image of several PAINT_OBJECTS and RGB_IMAGE: RGB_IMAGE from three channels is made first, then is used as INPUT for PAINT_OBJECTS where first nuclei are painted as contours (purple) and then cell bodies, also as contours (red).

There is modality in PAINT_OBJECTS used for creating masks based in objects. First, you don´t need to specify any input image, instead, used the reserved word:

- BLANK_BINARY if you want to use a binary image (black and white). This is useful for creating masks. This resultant image is 8 bits with 1 Channel. Image is completely black and loaded objects will be painted in the black canvas. Use WHITE as color for filling the object.
- BLANK_COLOUR if you want to store the image as 8 bits 3 channels. Functionality is the same as BLANK_BINARY, but now you can use color in objects.

CAUTION: If this option is used, parameters WIDTH and HEIGHT are used for set up the black image used as canvas for the objects. Be sure is the same size as the images you are working or you will get an error.

NOTE: Maybe looks unnecessary to fill these parameters, but next versions of Hcell will extend this capability to store selected ROIs from an image.

```
<MODULE seq="004a" name="FILE PROCESSING">
        <FUNCTION name="PAINT OBJECTS">
                <ASHOW>FALSE</ASHOW>
                <WIDTH>2048</WIDTH>
                <HEIGHT>2048</HEIGHT>
                <WINDOWSNAME>BINARY</WINDOWSNAME>
                <INPUT>BLANK_BINARY</INPUT>
                <COLOUR>WHITE</COLOUR>
                <THICKNESS>0</THICKNESS>
                <LOAD_OBJECTS>LOBS_OBJ2</LOAD_OBJECTS>
                <PAINT_CONTOURS>FALSE</PAINT_CONTOURS>
                <OUTPUT>LOBS_PAINTED</OUTPUT>
```

```
            </FUNCTION>
</MODULE>
```

### 1.1.8 SHOW

| SHOW | Possible Values | Default assignment |
|---|---|---|
| INPUT | Input image that is going to be shown. | “” |
| WINDOWSNAME | Windows name when image is shown. | “” |
| WIDTH | Window width | 800 |
| HEIGHT | Window height | 600 |
| TYPE | Not used | - |

```
  <MODULE seq="003" name="FILE PROCESSING">
            <FUNCTION name="SHOW">
                  <INPUT>NUCLEI_NORMALIZED</INPUT>
                  <WINDOWSNAME>NUCLEI</WINDOWSNAME>
                  <WIDTH>800</WIDTH>
                  <HEIGHT>600</HEIGHT>
                  <TYPE>WINDOW</TYPE>
            </FUNCTION>
      </MODULE>
```

NOTE: Type is not yet implemented. Future versions of Type will include WINDOW or RASTER, where raster could be a browser image using Javascript commands.

The checker system will not allow a bigger resolution than the one used in the by default monitor. Future versions will change this for secondary monitors.

### 1.1.9 PRINT

Use this if you want to show any message using the console. Messages shown here are independent from verbose mode.

| PRINT | Possible Values | Default assignment |
|---|---|---|
| MESSAGE | Any string | “” |

```
<MODULE seq="00Pm" name="FILE PROCESSING">
      <FUNCTION name="PRINT">
            <MESSAGE>###HELLO WORLD!###################</MESSAGE>
      </FUNCTION>
</MODULE>
```

### 1.1.10 CONVERT

Use this if you want to show any message using the console. Messages shown here are independent from verbose mode.

| CONVERT | Possible Values | Default assignment |
|---|---|---|
| INPUT | Input image that is going to be converted | “” |
| OUTPUT | Converted image | “” |
| DEPTH | 8,16, 32 or 64 bits | 8 |
| TYPE | RGB or GRAY | GRAY |

ATTENTION: Convert at your own risk! Conversion is not always checked between functions. You can get errors during execution if the input of a function is different from the depth expected. For example, some functions only accept 8 bits as input.

```
<MODULE seq="001" name="FILE PROCESSING">
      <FUNCTION name="CONVERT">
            <INPUT>INP</INPUT>
            <OUTPUT>OUT</OUTPUT>
            <DEPTH>8</DEPTH>
            <TYPE>RGB</TYPE>
      </FUNCTION>
</MODULE>
```

## 1.2 Image Processing Module

### 1.2.1 THRESHOLD

| THRESHOLD | Type and description | Default assignment |
|---|---|---|
| ASHOW | Shows a window with the result. | FALSE |
| INPUT | String naming the source image. | “” |
| THRESHOLD | Value with the threshold level of cut off. | “” |
| MAX_THRESHOLD | Maximum value used as reference for the cut off. Could be 255, 65534, 1 as standard for 8, 16 bits or normalized data.<br><br>E.g. If my image is normalized and my threshold is 0.5, my maximum would be 1.<br>I can use MAX to be my maximum intensity 0.8 and everything after 0.8 would be considered white. | 1 |

| | | |
|---|---|---|
| THRESHOLD_TYPE | String with one of the next values:<br>"**THRESH_BINARY"**<br>Normal threshold<br>If the intensity of the pixel is higher than threshold, then the new pixel intensity is set to a max_threshold. Otherwise, the pixels are set to 0.<br><br>**"THRESH_TRUNC"**<br>The maximum intensity value for the pixels is the threshold, if is greater, then its value is *truncated*, becames the threshold value.<br><br>**"THRESH_BINARY_INV"**<br>If the intensity of the pixel is higher than threshold, then the new pixel intensity is set to a 0. Otherwise, it is set to max_threshold.<br><br>**"THRESH_TOZERO"**<br>If pixel value is **lower** than thresh, the new pixel value will be set to 0.<br><br>**"THRESH_TOZERO_INV"**<br>If pixel is **greater** than threshold, the new pixel value will be set to 0.<br><br>**"THRESH_OTSU"**<br>It acts same as binary thresholding but applying Otsu´s algorithm (which takes the variance of the histogram in account for automatically adjust the threshold).<br>**Only 8 bit images** can be used in otsu and the **threshold value could be anything (it would not be taken in account).** Leave at 0. | "THRESH_ BINARY" |
| WINDOWSNAME | String with the windows name. | "" |
| OUTPUT | Name of image destination. Output is an 8 bit binary image with values between 255 and 0. | |

```
<MODULE seq="002a" name="IMAGE PROCESSING">
            <FUNCTION name="THRESHOLD">
                  <INPUT>IMAGE3_CY3</INPUT>
                  <THRESHOLD>0.0003</THRESHOLD>
                  <THRESHOLD_TYPE>THRESH_BINARY</THRESHOLD_TYPE>
                  <MAX_THRESHOLD>1</MAX_THRESHOLD>
                  <ASHOW>FALSE</ASHOW>
                  <WINDOWSNAME>THRESHOLD</WINDOWSNAME>
                  <OUTPUT>CY3_GTHRESH</OUTPUT>
            </FUNCTION>
</MODULE>
```

Reference: http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html

### 1.2.2 ADAPTIVE THRESHOLD

| ADAPTIVE THRESHOLD | Type and description | Default assignment |
|---|---|---|
| ASHOW | Boolean- show window after function application. | FALSE |
| INPUT | String – source image | - |
| OUTPUT | String – output image | - |
| THRESHOLD_TYPE | BINARY<br>Adaptive thresholding for 8 bit images, highly optimized. If your requirements are not high, this option is faster.<br><br>BINARY_INV.<br>Same as binary but returns the inverted image.<br><br>THRESH_16.<br>Adaptive thresholding. If you are working with 16 bit images and you don´t want to lose small details, use this thresholding option. CONSTANT is the value that is added to the threshold. Also known as Mean method where:<br><br>pixel = ( pixel > mean - c ) ? object : background | BINARY |
| WINDOWSNAME | Name of the ASHOW window. | “” |
| METHOD | GAUSSIAN or MEAN<br>Only for BINARY and BINARY_INV | GAUSSIAN |
| BLOCKSIZE | Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on. | 5 |
| CONSTANT | Constant subtracted from the mean or weighted mean. Normally, it is positive but may be zero or negative as well. | 0.0 |

```
<MODULE seq="002a" name="IMAGE PROCESSING">
            <FUNCTION name="ADAPTIVE THRESHOLD">
                  <INPUT>NUCLEI_NORMALIZED</INPUT>
                  <THRESHOLD_TYPE>THRESH_BINARY_16</THRESHOLD_TYPE>
                  <METHOD>GAUSSIAN</METHOD>
                  <CONSTANT>0.01</CONSTANT>
                  <BLOCKSIZE>151</BLOCKSIZE>
                  <ASHOW>FALSE</ASHOW>
                  <WINDOWSNAME>THRESHOLD</WINDOWSNAME>
                  <OUTPUT>NUCLEI_THRESH</OUTPUT>
            </FUNCTION>
      </MODULE>
```

MEAN Threshold value is a mean of BLOCKSIZE X BLOCKSIZE neighborhood of each pixel minus C.
GAUSSSIAN threshold value is a weighted sum (cross-correlation with a Gaussian window of BLOCKSIZE X BLOCKSIZE neighborhood of each each pixel minus C. The standard deviation is computed as `0.3*((BLOCKSIZE-1)*0.5 - 1) + 0.8`.

For the BINARY and BINARY_INV check documentation on
http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html

### 1.2.3 MAKE BRUSH

| MAKE BRUSH | Possible Values | Default assignment |
|---|---|---|
| ASHOW | Boolean- show window after function application. | |
| WINDOWSNAME | Name of the ASHOW window. | |
| BRUSH_SIZE_X | Size in pixels of our brush for the x axis. | 3 |
| BRUSH_SIZE_Y | Size in pixels of our brush for the y axis. | 3 |
| BRUSH_TYPE | A binary matrix forming a shape.<br>**GAUSSIAN_SHAPE**<br>**DIAMOND_SHAPE**<br>**BOX_SHAPE**<br>**CROSS_SHAPE**<br>**ELLIPSE_SHAPE** | GAUSSIAN_SHAPE |
| SIGMA | Standard Deviation value in case of using GAUSSIAN_SHAPE brush. It regulates the gaussian distribution shape (wider or narrower). | 1 |
| OUTPUT | Output name of brush | |

```
<MODULE seq="004a" name="IMAGE PROCESSING">
      <FUNCTION name="MAKE BRUSH">
            <ASHOW>FALSE</ASHOW>
            <WINDOWSNAME>MY BRUSH</WINDOWSNAME>
            <BRUSH_SIZE_X>51</BRUSH_SIZE_X>
            <BRUSH_SIZE_Y>51</BRUSH_SIZE_Y>
            <BRUSH_TYPE>GAUSSIAN_SHAPE</BRUSH_TYPE>
            <SIGMA>1</SIGMA>
            <OUTPUT>BRUSH NUMBER 1</OUTPUT>
      </FUNCTION>
</MODULE>
```

This function creates a structuring element that can be stored and used afterwards in morphological operators or in 2D filters.

Default position of brush is anchored at the center of the matrix (Only important for Cross shaped brushes).

NOTE: Be aware that filters are stored only once in the pool. Morphological operators, 2D filters and Numerical Operators (Scalar operators, Matrix to Matrix operators), they will be applied only once per pipeline execution.

### 1.2.4 ERODE

Erodes an image by using a specific structuring element. The function erodes the source image using the specified brush that determines the shape of a pixel neighborhood over which the minimum is taken.

| ERODE | Type and description | Default assignment |
|---|---|---|
| ASHOW | Boolean- show window after function application. | FALSE |
| INPUT | Input image; depth should be 8, 16 or 32. | “” |
| ITERATIONS | Number of times erosion is applied. | 1 |
| WINDOWSNAME | Name of the ASHOW window. | “” |
| OUTPUT | Output image of the same size and type as input. | “” |
| BRUSH | Brush name created previously with the function **Make brush**. | “” |

```
<MODULE seq="002c" name="IMAGE PROCESSING">
            <FUNCTION name="ERODE">
                  <ASHOW>FALSE</ASHOW>
                  <WINDOWSNAME>EROSION</WINDOWSNAME>
                  <INPUT>NUCLEI_THRESH</INPUT>
                  <BRUSH>BRUSH NUMBER 1</BRUSH>
                  <ITERATIONS>1</ITERATIONS>
                  <OUTPUT>NUCLEI_HIGHLY_ERODED</OUTPUT>
            </FUNCTION>
      </MODULE>
```

### 1.2.5 DILATE

Dilates an image by using a specific structuring element. The function dilates the source image using the specified brush that determines the shape of a pixel neighborhood over which the maximum is taken.

| DILATE | Type and description | Default assignment |
|---|---|---|
| ASHOW | Boolean- show window after function application. | FALSE |
| INPUT | Input image; depth should be 8, 16 or 32. | “” |
| ITERATIONS | Number of times dilation is applied. | 1 |
| WINDOWSNAME | Name of the ASHOW window. | “” |
| OUTPUT | Output image of the same size and type as input. | “” |
| BRUSH | Brush name created previously with the function **Make brush**. | “” |

```
<MODULE seq="008e" name="IMAGE PROCESSING">
      <FUNCTION name="DILATE">
            <ASHOW>FALSE</ASHOW>
            <WINDOWSNAME>DILATE</WINDOWSNAME>
            <INPUT>NUCLEI_ERODED</INPUT>
            <BRUSH>BRUSH NUMBER 4</BRUSH>
            <ITERATIONS>1</ITERATIONS>
            <OUTPUT>NUCLEI_DILATED</OUTPUT>
      </FUNCTION>
</MODULE>
```

### 1.2.6 MORPHOLOGICAL OPERATION

Performs advanced morphological transformations.

| DILATE | Type and description | Default assignment |
|---|---|---|
| ASHOW | Boolean- show window after function application. | FALSE |
| INPUT | Input image; depth should be 8, 16 or 32. | “” |
| OPERATION | Type of a morphological operation that can be one of the following:<br>**OPENING**<br>**CLOSING**<br>**GRADIENT**<br>**TOP HAT**<br>**BLACK HAT** | “” |
| ITERATIONS | Number of times operation is applied. | 1 |
| WINDOWSNAME | Name of the ASHOW window. | “” |

| | | |
|---|---|---|
| OUTPUT | Output image of the same size and type as input. | “” |
| BRUSH | Brush name created previously with the function **Make brush**. | “” |

```
<MODULE seq="004a" name="IMAGE PROCESSING">
      <FUNCTION name="MAKE BRUSH">
            <ASHOW>FALSE</ASHOW>
            <WINDOWSNAME>MY BRUSH</WINDOWSNAME>
            <BRUSH_SIZE_X>1</BRUSH_SIZE_X>
            <BRUSH_SIZE_Y>1</BRUSH_SIZE_Y>
            <BRUSH_TYPE>BOX_SHAPE</BRUSH_TYPE>
            <SIGMA>0</SIGMA>
            <OUTPUT>BRUSH NUMBER 1</OUTPUT>
      </FUNCTION>
</MODULE>

<MODULE seq="007c" name="IMAGE PROCESSING">
      <FUNCTION name="MORPHOLOGICAL OPERATION">
            <ASHOW>FALSE</ASHOW>
            <WINDOWSNAME>OPENING</WINDOWSNAME>
            <OPERATION>OPENING</OPERATION>
            <ITERATIONS>1</ITERATIONS>
            <INPUT>NUC_TRESH</INPUT>
            <BRUSH> BRUSH NUMBER 1</BRUSH>
            <OUTPUT>NUC_OPENING</OUTPUT>
      </FUNCTION>
</MODULE>
```

### 1.2.7 BOUNDARY EXTRACTION

Boundaries using morphological operators, applies erosion of a chosen thickness and returns the contour. It doesn´t store the contours as objects, returns an image with the contours.

| BOUNDARY EXTRACTION | Type and description | Default assignment |
|---|---|---|
| ASHOW | Boolean- show window after function application. | FALSE |
| THICKNESS | Integer - line thickness of the resulting contours. | |
| COLOUR | Boundary color of representation. Check Paint Objects for more information. RANDOM is not implemented in this function. | |
| INPUT | Input image. | |
| WINDOWSNAME | Name of the ASHOW window. | |

| | | |
|---|---|---|
| OUTPUT | Output image of 8 bit binary, or 8 bit 3 channels if Colour is different from WHITE. | |

```
<MODULE seq="006" name="IMAGE PROCESSING">
      <FUNCTION name="BOUNDARY EXTRACTION">
            <ASHOW>TRUE</ASHOW>
            <WINDOWSNAME>B_EXTRACTION</WINDOWSNAME>
            <THICKNESS>3</THICKNESS>
            <COLOUR>RED</COLOUR>
            <INPUT>IMAGE4</INPUT>
            <OUTPUT>IMAGE5</OUTPUT>
      </FUNCTION>
</MODULE>
```

Check Example 5.

### 1.2.8 FIND CONTOURS

Detects contours of objects and saves them as objects.

| FIND CONTOURS | Possible Values | Default assignment |
|---|---|---|
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | |
| THICKNESS | Integer - line thickness of the resulting contours. | |
| COLOUR | Boundary color of representation. Check Paint Objects for more information. | |
| INPUT | Input image. | |
| OUTPUT | Output image 8 bit 3 channels. | |
| SAVE_OBJECTS | Name of list of objects detected. | |
| APPROXIMATION | **NONE**<br>**SIMPLE**<br>**TEH CHIN 1**<br>**TEH CHIN 2** | |
| RETRIEVAL MODE | **EXTERNAL**<br>**LIST**<br>**CONNECTED COMPONENTS**<br>**TREE** | |

```
<MODULE seq="006" name="IMAGE PROCESSING">
      <FUNCTION name="FIND CONTOURS">
            <ASHOW>TRUE</ASHOW>
            <WINDOWSNAME>CONTOURS</WINDOWSNAME>
            <COLOUR>RANDOM</COLOUR>
            <THICKNESS>1</THICKNESS>
            <RETRIEVAL_MODE>EXTERNAL</RETRIEVAL_MODE>
            <APPROXIMATION>NONE</APPROXIMATION>
            <INPUT>IMAGE4</INPUT>
            <OUTPUT>IMAGE5</OUTPUT>
            <SAVE_OBJECTS>CONT_1</SAVE_OBJECTS>
      </FUNCTION>
</MODULE>
```

Contour retrieval mode:
EXTERNAL retrieves only the extreme outer contours
LIST retrieves all of the contours without establishing any hierarchical relationships.
CCOMP retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
TREE retrieves all of the contours and reconstructs a full hierarchy of nested contours

Contour approximation method

NONE stores absolutely all the contour points.
SIMPLE compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.
Teh-Chin 1 and 2 applies one of the flavors of the Teh-Chin chain approximation algorithm.
See ***Teh, C.H. and Chin, R.T.,* On the Detection of Dominant Points on Digital Curve. *PAMI 11 8, pp 859-872 (1989)*** for details.

For more information about retrieval and approximation mode, go to :
http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=drawcontours#cv.DrawContours

### 1.2.9 FILLHULL

Fill holes from a closed structure.

| FILLHULL | Possible Values | Default assignment |
|---|---|---|
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | |
| INPUT | Source image. Must be 8 bit 1 channel. | |
| OUTPUT | Destiny Image. Output is 8 bit 1 channel | |

```
<MODULE seq="008b" name="IMAGE PROCESSING">
```

```
        <FUNCTION name="FILLHULL">
        <ASHOW>FALSE</ASHOW>
        <WINDOWSNAME>Filling coordinates</WINDOWSNAME>
        <INPUT>NUC_TRESH_HOLES</INPUT>
        <OUTPUT>NUC_TRESH_HOLES_FILLED</OUTPUT>
        </FUNCTION>
</MODULE>
```

### 1.2.10 CANNY EDGES

Canny edge detector. First the image is smoothed by Gaussian convolution. Then a simple 2-D first derivative operator is applied to the smoothed image to highlight regions of the image with high first spatial derivatives. Edges give rise to ridges in the gradient magnitude image. The algorithm then tracks along the top of these ridges and sets to zero all pixels that are not actually on the ridge top so as to give a thin line in the output, a process known as non-maximal suppression. The tracking process exhibits hysteresis controlled by two thresholds: T1 high and T2 low, with T1 > T2. Tracking can only begin at a point on a ridge higher than T1. Tracking then continues in both directions out from that point until the height of the ridge falls below T2. This hysteresis helps to ensure that noisy edges are not broken up into multiple edge fragments.

The effect of the Canny operator is determined by three parameters --- the width of the Gaussian kernel used in the smoothing phase, and the upper and lower thresholds used by the tracker.
Increasing the width of the Gaussian kernel reduces the detector's sensitivity to noise, at the expense of losing some of the finer detail in the image. The localization error in the detected edges also increases slightly as the Gaussian width is increased.
Usually, the upper tracking threshold can be set quite high, and the lower threshold quite low for good results. Setting the lower threshold too high will cause noisy edges to break up. Setting the upper threshold too low increases the number of spurious and undesirable edge fragments appearing in the output.

| CANNY | Possible Values | Default assignment |
|---|---|---|
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |
| HIGH_THRESHOLD | First threshold for the hysteresis procedure. | 0 |
| INPUT | Single-channel 8-bit input image. | “” |
| KERNEL_SIZE | Aperture size smoothing operator. | 3 |

| | | |
|---|---|---|
| L2_GRADIENT | A flag, indicating whether a more accurate $L_2$ norm $= \sqrt{(dI/dx)^2 + (dI/dy)^2}$ should be used to calculate the image gradient magnitude ( L2gradient=true ), or whether the default $L_1$ norm $= \lvert dI/dx \rvert + \lvert dI/dy \rvert$ is enough ( L2gradient=false ). | FALSE |
| LOW_THRESHOLD | Second threshold for the hysteresis procedure. | 0 |
| OUTPUT | Output edge map; it has the same size and type as input image . | "" |

```
<MODULE seq="004" name="IMAGE PROCESSING">
      <FUNCTION name="CANNY">
            <ASHOW>TRUE</ASHOW>
            <WINDOWSNAME>CANNY EDGES</WINDOWSNAME>
            <HIGH_THRESHOLD>10</HIGH_THRESHOLD>
            <LOW_THRESHOLD>100</LOW_THRESHOLD>
            <L2_GRADIENT>FALSE</L2_GRADIENT>
            <KERNEL_SIZE>3</KERNEL_SIZE>
            <INPUT>IMAGE2</INPUT>
            <OUTPUT>IMAGE3</OUTPUT>
      </FUNCTION>
</MODULE>
```

The value `LOW THRESHOLD` is used for edge linking. The largest value `HIGH_THRESHOLD` is used to find initial segments of strong edges. For more information you can check:
http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html
http://en.wikipedia.org/wiki/Canny_edge_detector
http://homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm

### 1.2.11 LAPLACIAN

| LAPLACIAN | Possible Values | Default assignment |
|---|---|---|
| INPUT | Source image. | "" |
| OUTPUT | Destination image of the same size and the same number of channels as input | "" |
| KERNEL_SIZE | Aperture size used to compute the second-derivative filters. The size must be positive and odd. | 3 |
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |

The function calculates the Laplacian of the source image by adding up the second x and y derivatives calculated using the Sobel operator:

$$\texttt{dst} = \Delta \texttt{src} = \frac{\partial^2 \texttt{src}}{\partial x^2} + \frac{\partial^2 \texttt{src}}{\partial y^2}$$

This is done when ksize > 1 . When ksize == 1 , the Laplacian is computed by filtering the image with the following $3 \times 3$ aperture:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

```
<MODULE seq="006" name="IMAGE PROCESSING">
      <FUNCTION name="LAPLACIAN">
            <ASHOW>TRUE</ASHOW>
            <WINDOWSNAME>LAPLACIAN</WINDOWSNAME>
            <KERNEL_SIZE>3</KERNEL_SIZE>
            <INPUT>IMAGE5</INPUT>
            <OUTPUT>IMAGE6</OUTPUT>
      </FUNCTION>
</MODULE>
```

See example 8. Reference:
http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/laplace_operator/laplace_operator.html
http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

### 1.2.12 **GAUSSIAN BLUR**

Blurs an image using a Gaussian filter.

| GAUSSIAN BLUR | Possible Values | Default assignment |
|---|---|---|
| INPUT | Source image. | “” |
| OUTPUT | Destination image of the same size and the same number of channels as input | “” |
| HEIGHT | Kernel size in Y direction. | 3 |
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |
| WIDTH | Kernel size in X direction. | 3 |
| SIGMA_X | Gaussian kernel standard deviation in X direction. | 1 |

| | | |
|---|---|---|
| SIGMA_Y | Gaussian kernel standard deviation in Y direction.<br>If 0, is equal to X direction. | 0 |

```
<MODULE seq="006" name="IMAGE PROCESSING">
      <FUNCTION name="GAUSSIAN BLUR">
            <ASHOW>TRUE</ASHOW>
            <WINDOWSNAME>GBLUR</WINDOWSNAME>
            <WIDTH>3</WIDTH>
            <HEIGHT>3</HEIGHT>
            <SIGMA_X>0</SIGMA_X>
            <SIGMA_Y>0</SIGMA_Y>
            <INPUT>IMAGE4</INPUT>
            <OUTPUT>IMAGE5</OUTPUT>
      </FUNCTION>
</MODULE>
```

See example 8. Also, for more information see:

http://docs.opencv.org/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html

### 1.2.13 **SOBEL**

| SOBEL | Possible Values | Default assignment |
|---|---|---|
| INPUT | Source image. | “” |
| OUTPUT | Destination image of the same size and the same number of channels as input | “” |
| OPERATION | SOBEL or SCHARR. Explanation below. | SOBEL |
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |
| SIZE | Kernel size | 3 |
| X_ORDER | Order of the derivative x. | 1 |
| Y_ORDER | Order of the derivative y. | 1 |

```
</MODULE>
      <MODULE seq="006" name="IMAGE PROCESSING">
      <FUNCTION name="SOBEL">
            <ASHOW>TRUE</ASHOW>
            <WINDOWSNAME>SOBEL</WINDOWSNAME>
            <OPERATION>SOBEL</OPERATION>
            <X_ORDER>1</X_ORDER>
```

```
            <Y_ORDER>1</Y_ORDER>
            <SIZE>3</SIZE>
            <INPUT>IMAGE5</INPUT>
            <OUTPUT>IMAGE6</OUTPUT>
        </FUNCTION>
    </MODULE>
```

In all cases except one, a SIZExSIZE kernel is used to calculate the derivative. When SIZE=1 , the 3x1 or 1x3 kernel is used (that is, no Gaussian smoothing is done). SIZE = 1 can only be used for the first or the second x- or y- derivatives.

There is also the special value OPERATION = SCHARR that corresponds to the 3x3 Scharr filter that may give more accurate results than the 3x3 Sobel. The Scharr aperture is

$$\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

for the x-derivative, or transposed for the y-derivative.

The function calculates an image derivative by convolving the image with the appropriate kernel:

$$Output = \frac{\partial^{xorder+yorder}\, Input}{\partial x^{xorder}\, \partial y^{yorder}}$$

The Sobel operators combine Gaussian smoothing and differentiation, so the result is more or less resistant to the noise.

More information in:
http://en.wikipedia.org/wiki/Sobel_operator
http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html
http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

See Example 6.

### 1.2.14 **BLUR**

| **BLUR** | **Possible Values** | **Default assignment** |
|---|---|---|
| INPUT | Source image. | “” |
| OUTPUT | Destination image of the same size and the same number of channels as input | “” |
| OPERATION | BOX, MEDIAN or DESPECKLE | BOX |
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |
| SIZE | Kernel size | 3 |
| HEIGHT | Order of the derivative x. | 1 |

| | | |
|---|---|---|
| WIDTH | Order of the derivative y. | 1 |

Using type BOX The function smoothes an image using the normalized box kernel:

$$K = \frac{1}{\text{Height x Width}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ & & \cdots\cdots & & & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

MEDIAN uses a median filter.
DESPECKLE removes the speckle noise using a 3x3 median filter. It replaces each pixel with the median value in its 3 x 3 neighborhood. Median filters a good at removing salt and pepper noise but blurring the image in return.

http://docs.opencv.org/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html
http://en.wikipedia.org/wiki/Median_filter

```
<MODULE seq="006" name="IMAGE PROCESSING">
      <FUNCTION name="BLUR">
            <ASHOW>TRUE</ASHOW>
            <WINDOWSNAME>BLUR</WINDOWSNAME>
            <WIDTH>5</WIDTH>
            <HEIGHT>5</HEIGHT>
            <INPUT>IMAGE4</INPUT>
            <OUTPUT>IMAGE5</OUTPUT>
            <!-- Use BOX or DESPECKLE -->
            <OPERATION>MEDIAN</OPERATION>
      </FUNCTION>
</MODULE>
```

### 1.2.15 **CREATE KERNEL**

Creates a matrix that can be stored and used as kernel for filtering operations.

| CREATE KERNEL | Possible Values | Default assignment |
|---|---|---|
| OUTPUT | Kernel name to be stored. | “” |
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | “” |
| KERNEL_SIZE | Kernel size. Our kernel is a KERNEL_SIZE x KERNEL_SIZE matrix with the values of matrix. | 3 |

| | | |
|---|---|---|
| MATRIX | A floating point matrix.<br>The matrix is according to the kernel size, read as series of integer numbers in the format<br><br>1,2,3,4;<br><br>For each row.<br>For identity use the keyword EYE.<br>For ones use the keyword ONES.<br>For zeros use the keyword ZEROS.<br>You also can initialize the matrix to any number of your choice, writing the name alone:<br><MATRIX>5.0</MATRIX> | All ones<br>111<br>111<br>111 |
| DFACTOR or FACTOR | Dividing factor as a double. All the matrix numbers will get divided by this number. 1 by default.<br><br>KERNEL = 1/DFACTOR * MATRIX<br><br>Use FACTOR if you have decided to use a value from a previous operation. | 1.0 |

```
<MODULE seq="004" name="IMAGE PROCESSING">
      <FUNCTION name="CREATE KERNEL">
            <ASHOW>FALSE</ASHOW>
            <WINDOWSNAME>NULL</WINDOWSNAME>
            <KERNEL_SIZE>3</KERNEL_SIZE>
            <DFACTOR>1</DFACTOR>
            <!--sharpen kernel example-->
            <MATRIX>
                        0,-1,0;
                        -1,5,-1;
                        0,-1,0
            </MATRIX>
            <OUTPUT>KERNEL 1</OUTPUT>
      </FUNCTION>
</MODULE>
```

See Example 9.

### 1.2.16 **FILTER2D**

The kernel is convolved along the input image generating a new output image.

| FILTER2D | Possible Values | Default assignment |
|---|---|---|
| INPUT | Source image. | “” |

| | | |
|---|---|---|
| OUTPUT | Destination image of the same size and the same number of channels as input. | “” |
| KERNEL_NAME | Kernel coming from a kernel instruction. | None, must be provided. |
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |

```
<MODULE seq="005" name="IMAGE PROCESSING">
      <FUNCTION name="FILTER2D">
            <ASHOW>TRUE</ASHOW>
            <WINDOWSNAME>KERNEL</WINDOWSNAME>
            <INPUT>IMAGE2</INPUT>
            <KERNEL_NAME>KERNEL 1</KERNEL_NAME>
            <OUTPUT>IMAGE4</OUTPUT>
      </FUNCTION>
</MODULE>
```

See example 9.

### 1.2.17 **FLOODFILL**

Given a set of seeds, applies region growing flooding with the same colour the region where the point is. Is an automated version of the bucket from any image editor.

| **FLOODFILL** | **Possible Values** | **Default assignment** |
|---|---|---|
| SEEDS | Starting points for filling. | “” |
| LOW_DIFF | Maximal lower brightness/color difference between the currently observed pixel and one of its neighbors belonging to the component, or a seed pixel being added to the component. | 0 |
| UP_DIFF | Maximal upper brightness/color difference between the currently observed pixel and one of its neighbors belonging to the component, or a seed pixel being added to the component. | 0 |
| INPUT | Source image. | “” |
| OUTPUT | Destination image of the same size as input, but RGB (3 channels): | “” |

| | | |
|---|---|---|
| COLOUR | Color of the filling region. Check Paint Objects for more information. | BLACK |
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |

```
<MODULE seq="006" name="IMAGE PROCESSING">
      <FUNCTION name="FLOODFILL">
      <ASHOW>TRUE</ASHOW>
            <WINDOWSNAME>Filling coordinates</WINDOWSNAME>
            <COLOUR>RANDOM</COLOUR>
            <SEEDS>OBJECTS</SEEDS>
            <LOW_DIFF>0</LOW_DIFF>
            <UP_DIFF>0</UP_DIFF>
            <INPUT>IMAGE3</INPUT>
            <OUTPUT>IMAGE6</OUTPUT>
      </FUNCTION>
</MODULE>
```

More info in
http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#floodfill

### 1.2.18 **SCALAR OPERATION**

| SCALAR OPERATION | Possible Values | Default assignment |
|---|---|---|
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |
| INPUT | Source matrix (image or kernel). | “” |
| OUTPUT | Destination matrix (image or kernel). | “” |

| | | |
|---|---|---|
| OPERATIONS | SUM – Sums FACTOR to each element in the matrix.<br>DIFF – Subtracts FACTOR to each element in the matrix.<br>DIV – Divides each element from the matrix by FACTOR.<br>MULT – Multiplies each element from the matrix by FACTOR.<br>POWER – Applies the power operation to the matrix, to the power FACTOR.<br>INV – Inverses the matrix. In this case factor is not taken in account, must be left at 0. | INV |
| FACTOR | Double used for the operation. | 0 |
| ITERATIONS | Number of times operation is performed over the resultant matrix. | 1 |

```
<MODULE seq="005a" name="IMAGE PROCESSING">
      <FUNCTION name="SCALAR OPERATION">
            <ASHOW>FALSE</ASHOW>
            <WINDOWSNAME>BRUSH NUMBER 1</WINDOWSNAME>
            <INPUT>BRUSH NUMBER 1</INPUT>
            <OPERATION>DIV</OPERATION>
            <ITERATIONS>1</ITERATIONS>
            <FACTOR>12.91571</FACTOR>
            <OUTPUT>BRUSH NUMBER M1</OUTPUT>
      </FUNCTION>
</MODULE>
```

### 1.2.19 **MATRIX OPERATION**

Takes 2 inputs matrices and applies a mathematical chosen operator.

| MATRIX OPERATION | Possible Values | Default assignment |
|---|---|---|
| INPUT1 | Image or kernel source. | “” |
| INPUT2 | Image or kernel source. Must have same size as INPUT1 | “” |
| ITERATIONS | Integer, number of times operation is repeated, taking INPUT2 as a constant element.<br>Ex. Iterations = 2<br>(A+B)+B<br>Iterations = 3<br>(((A+B)+B)+B) | 1 |

| | | |
|---|---|---|
| OPERATION | Given input matrices A and B (input1 and input2 respectively) :<br>MOR A or B<br>MXOR A xor B<br>MAND A and B<br>MSUM A + B<br>MDIFF A – B<br>MDIV A / B per element<br>MMULT A * B per element<br>MMULT_P A*B product | MOR |
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |
| OUTPUT | Resultant matrix. | “” |

Example:

```
<MODULE seq="009e" name="IMAGE PROCESSING">
      <FUNCTION name="MATRIX OPERATION">
            <ASHOW>FALSE</ASHOW>
            <WINDOWSNAME>XOR</WINDOWSNAME>
            <INPUT1>CY3_OPENING</INPUT1>
            <INPUT2>CY3_OPENING2</INPUT2>
            <OPERATION>MOR</OPERATION>
            <ITERATIONS>1</ITERATIONS>
            <OUTPUT>CYTOPLASM_REGIONS</OUTPUT>
      </FUNCTION>
</MODULE>
```

### 1.2.20 **Matrix to Scalar operation**

Applies an operator to a matrix which extracts a factor, like mean, sum or trace.

| **MATRIX TO SCALAR OPERATION** | **Possible Values** | **Default assignment** |
|---|---|---|
| INPUT | Source image. | “” |

| | | |
|---|---|---|
| OPERATION | Mathematical operation applied to the image as a matrix of numbers.<br>**Sum:** SUM sum of all elements.<br>**Mean:** MEAN mean of all elements.<br>**Trace:** TRACE trace of the matrix image.<br>**Max :** MAX maximum element of the matrix<br>**Min:** MIN minimum element of the matrix | MTSMEAN |
| OUTPUT_FACTOR | Scalar value after applying the operation. The name can´t start by a number. | “” |

**ATTENTION :** Factors extracted from this function **can only be used** in functions **SCALAR OPERATION and CREATE KERNEL** using the parameter FACTOR.

```
<MODULE seq="013a" name="IMAGE PROCESSING">
      <FUNCTION name="MATRIX TO SCALAR OPERATION">
            <INPUT>IMAGE2_DAPI</INPUT>
            <!-- MEAN, SUM, TRACE, SD from a matrix-->
            <OPERATION>MEAN</OPERATION>
            <OUTPUT_FACTOR>MEAN_1</OUTPUT_FACTOR>
      </FUNCTION>
</MODULE>
```

Equivalent to Factor = Operation(Input Matrix) where the result is a number, like the trace or the mean.

## 1.3 SEGMENTATION PROCESSING

### 1.3.1 Watershed

Watershed marker based algorithm. Requires foreground and background for masking. Before passing the image to the function, you have to roughly outline the desired regions in the image FOREGROUND with positive (>0) indices. The markers are seeds of the future image regions. BACKGROUND is a mask applied to all pixels that are going to be excluded.

| WATERSHED | Possible Values | Default assignment |
|---|---|---|
| INPUT | Source image: the image you want to segment. Must be binary 8-bit or RGB 8 bit. | “” |
| OUTPUT | Destination image of the same size and the same number of channels as input. | “” |
| FOREGROUND | Marker regions (seeds). | “” |
| BACKGROUND | Mask for removal pixels to not being evaluated. | “” |

| | | |
|---|---|---|
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |
| SAVE_OBJECTS | Objects segmented from Watershed. | “” |

```
<MODULE seq="009" name="SEGMENTATION PROCESSING">
      <FUNCTION name="WATERSHED">
            <FOREGROUND>NUCLEI_ERODED2</FOREGROUND>
            <BACKGROUND>NUCLEI_DILATED2</BACKGROUND>
            <INPUT>NUCLEI_NORMALIZED</INPUT>
            <OUTPUT>SEGMENTED_NUCLEI</OUTPUT>
            <SAVE_OBJECTS>CELLS_OBJ_FINAL</SAVE_OBJECTS>
            <ASHOW>FALSE</ASHOW>
            <WINDOWSNAME>SEGMENTATION by Watershed</WINDOWSNAME>
      </FUNCTION>
</MODULE>
```

More information in:
http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_watershed/py_watershed.html

### 1.3.2 Propagate

| PROPAGATE | Possible Values | Default assignment |
|---|---|---|
| SEEDS | Points for region growing. | |
| INPUT | Source original image. | |
| MASK | Source for a mask of 8 bits. | |
| SAVE_OBJ | Resultant objects. | |
| LAMBDA | Double value. | |

```
<MODULE seq="012" name="SEGMENTATION PROCESSING">
      <FUNCTION name="PROPAGATE">
            <SEEDS>NUC_OBJ</SEEDS>
            <INPUT>BODIES</INPUT>
            <MASK>MASK</MASK>
            <SAVE_OBJ>BODIES_OBJ</SAVE_OBJ>
            <LAMBDA>0.000002</LAMBDA>
      </FUNCTION>
</MODULE>
```

Algorithm based on the article:
http://www.cellprofiler.org/linked_files/Papers/JonesCVBIA2005.pdf

### 1.3.3 Label

Subsets of connected components are uniquely labeled.

| LABEL | Possible Values | Default assignment |
|---|---|---|
| INPUT | Source image, binary 8-bits. | “” |
| OUTPUT | Objects output from labelled components. | “” |

http://en.wikipedia.org/wiki/Connected-component_labeling

```
<MODULE seq="006" name="SEGMENTATION PROCESSING">
      <FUNCTION name="LABEL">
            <INPUT>NUC_END</INPUT>
            <OUTPUT>NUC_OBJ</OUTPUT>
      </FUNCTION>
</MODULE>
```

### 1.3.4 Split by watershed

Takes a binary mask from an images and apply the watershed algorithm for splitting big objects into smaller ones.

| SPLIT BY WATERSHED | Possible Values | Default assignment |
|---|---|---|
| INPUT | Source as a mask of 8 bits containing the objects that are going to be evaluated. | “” |
| EXT | Integer, value of extra pixels to consider from neighborhood. The bigger the EXT value is, the more neighbors are evaluated for the contour. | 0 |
| TOL | Double, Tolerance refers to tolerated distance. When watersheds are evaluated, distances between max and min step are checked. If the difference is smaller than tolerance, the closest distance is assigned. | 0.0 |
| REFERENCE | Original image with the gray values. | “” |
| SAVE_OBJ | Name of the objects detected after applying watershed. | “” |
| METHOD | Splitting method:<br>INTENSITY<br>DISTANCE | DISTANCE |

```
<MODULE seq="004b" name="SEGMENTATION PROCESSING">
<FUNCTION name="SPLIT BY WATERSHED">
      <INPUT>LOBS_PAINTED</INPUT>
```

```
            <REFERENCE>NUCLEI_NORMALIZED</REFERENCE>
            <!-- Methods DISTANCE, or INTENSITY -->
            <METHOD>DISTANCE</METHOD>
            <EXT>1</EXT>
            <TOL>1.0</TOL>
            <SAVE_OBJ>NUCS_WATERSHED</SAVE_OBJ>
      </FUNCTION>
</MODULE>
```

In the method parameter INTENSITY applies the watershed takes the intensity gray values (normalized between 0 and 1) as a probability value and multiplies it by the distance transform. It works good with cell nuclei, which are whiter in the center. DISTANCE uses the distance transform only.

### 1.3.5 Split by Otsu

This function analyzes every object loaded in load objects an applies locally to only that object Otsu thresholding. If the object is formed by multiple intensity peaks, will get split in different subset of objects. The new objects are stored in save objects.

Can be used to split clumps of nuclei cells to have seeds, that then can be applied to get the original number of nuclei.

| SPLIT BY OTSU | Possible Values | Default assignment |
|---|---|---|
| REFERENCE | Image with reference values used for thresholding | “” |
| LOAD_OBJECTS | Objects to evaluate from the reference image. | |
| SAVE_OBJECTS | Objects evaluated got split in a new set of objects if they are over the calculated otsu threshold. | |

```
      <MODULE seq="003c" name="SEGMENTATION PROCESSING">
      <FUNCTION name="SPLIT BY OTSU">
            <REFERENCE>NUCLEI_NORMALIZED</REFERENCE>
            <LOAD_OBJECTS>LOBS_OBJ</LOAD_OBJECTS>
            <SAVE_OBJECTS>NUCS_OTSU</SAVE_OBJECTS>
      </FUNCTION>
</MODULE>
```

### 1.3.6 Split by Fragmentation

This algorithm can be used in images where small objects can clump and be difficult to be separated from other big objects. We need to split previously the big objects using another algorithm.

| SPLIT BY FRAGMENTATION | Possible Values | Default assignment |
|---|---|---|
| REFERENCE | Image with reference values used for referencing the objects. | “” |
| BIG_OBJECTS | Objects to evaluate if they are susceptible of being fragmented. | “” |
| FRAG_OBJECTS | Objects fragmented using an algorithm. | “” |
| COEFFICIENT | Integer number, approximately the size of the object of interest. | 1 |
| SAVE_OBJECTS | Objects evaluated got split in a new set of objects. | “” |

More variation between intensity values in a local area happens when cells are clustering.

Given an Object A of size $S_A$ in pixels, and $N_{SA}$ is the total number of regions obtained by local segmentation of A, we define the fragmentation decision function as:

$$if\left(\frac{S_A}{expected\ area}\right) > N_{SA}\ \ is\ \ a\ clump$$
$$else\ is\ Large\ Object\ (Non-identified)$$

where the expected area in pixels is a parameter provided by the user about how big in pixels the expects size of each individual cell nucleus.
*Example:* Object A is 500 pixels. If it is formed by cell nuclei of 50 pixels each, it would be formed by at least, 10 cells. Now we apply our algorithm and we obtain 6 objects. Our threshold decision checks that 10 > 6, our object is not fragmented enough and then classified as a big object (possible outlier).

In practice, dense clumps for drosophila are difficult to split, with most of the nuclei overlapping with each other. In our case, the expect size of cell nucleus was increased to two times the expected value in order to find an optimal split solution. The boundary between cell clumps and other type of clusters is dependent on the type of cells and images, and thus this expected size value must be adjusted manually in the COEFFICIENT parameter.

```
<MODULE seq="004e" name="SEGMENTATION PROCESSING">
            <FUNCTION name="SPLIT BY FRAGMENTATION">
                        <REFERENCE>NUCLEI_NORMALIZED</REFERENCE>
                        <BIG_OBJECTS>LOBS_OBJ2</BIG_OBJECTS>
                        <FRAG_OBJECTS>NUCS_WATERSHED_2</FRAG_OBJECTS>
                        <COEFFICIENT>150</COEFFICIENT>
                        <SAVE_OBJECTS>NUCS_WATERSHED_3</SAVE_OBJECTS>
            </FUNCTION>
      </MODULE>
```

For example, Watershed or Split by Otsu can be used to fragment a set of objects, but some of those splits are wrong, they are simply big objects. We can use then split by fragmentation and get a new subset where the big objects that don´t pass our applied threshold are stored such as that.

### 1.3.7 Merge Objects

Given a reference image where different objects have been segmented, joins all the objects in one set.

| MERGE OBJECTS | Possible Values | Default assignment |
|---|---|---|
| REFERENCE | Image with reference values used for referencing the objects. | “” |
| LOAD_OBJECTS_1 | Loads the first set of segmented objects. | “” |
| LOAD_OBJECTS_2 | Loads the second set of segmented objects | “” |
| OPERATION | FUSION<br>When any object of OBJECTS_2 overlaps with one of the previous objects from OBJECTS_1, object from 1 and object from 2 become only one object.<br>FIRST_FOUND<br>When two objects overlap, objects from OBJECTS_1 have privilege and overlapping regions are kept as belonging to OBJECTS_1 and remove from the object belonging to OBJECTS_2.<br>FAST<br>Only recommended if you know that the objects don´t have overlapping regions. It is just doing a simple union of all objects, but if objects are overlapping or repeated, they will be also in the result. | FIRST_FOUND |
| OUTPUT | New set of objects, result of merging objects 1 and objects 2. | |

| | | |
|---|---|---|
| MASK | If we want to apply a binary mask, in which case, all pixels put to WHITE, i.e., to 0 are the only ones to being stored in the new set.<br>Value NO if you don´t want to use a mask.<br>It does not work with the FAST method. | |

```
<MODULE seq="004h" name="SEGMENTATION PROCESSING">
            <FUNCTION name="MERGE OBJECTS">
            <REFERENCE>NUCLEI_NORMALIZED</REFERENCE>
            <LOAD_OBJECTS_1>CELLS_OBJ3</LOAD_OBJECTS_1>
            <LOAD_OBJECTS_2>CELLS_OBJ4</LOAD_OBJECTS_2>
            <OPERATION>FUSION</OPERATION>
            <OUTPUT>CELLS_OBJ5</OUTPUT>
            <MASK>NO</MASK>
            </FUNCTION>
</MODULE>
```

### 1.3.8 Hough Transform

| HOUGH TRANSFORM | Possible Values | Default assignment |
|---|---|---|
| INPUT | Source image: the image you want to segment. Must be binary 8-bit or RGB 8 bit. | “” |
| LINES | Output vector of lines. Each line is represented by a 4-element vector $(x_1, y_1, x_2, y_2)$, where $(x_1, y_1)$ and $(x_2, y_2)$ are the ending points of each detected line segment. | “” |
| RHO | Distance resolution of the accumulator in pixels. | 1 |
| DTHETA | Angle resolution of the accumulator in degrees.<br>Double value. | 1 |
| THRESHOLD_INTERSECTIONS | Accumulator threshold parameter. Only those lines are returned that get enough votes ($> threshold$). | 100 |

| | | |
|---|---|---|
| PROBABILISTIC | | FALSE |
| MIN_LIN_LENGTH | Minimum line length. Line segments shorter than that are rejected. | 0 |
| MAX_LINE_GAP | Maximum allowed gap between points on the same line to link them. | 0 |
| COLOUR | Color of represented lines. All colors are allowed except RANDOM. | RED |
| ASHOW | Boolean- show window after function application. | FALSE |
| WINDOWSNAME | Name of the ASHOW window. | FALSE |

```
<MODULE seq="001" name="SEGMENTATION PROCESSING">
      <FUNCTION name="HOUGH TRANSFORM">
            <INPUT>IMAGE</INPUT>
            <LINES>LINES</LINES>
            <RHO>1</RHO>
            <THETA>1</THETA>
           <THRESHOLD_INTERSECTIONS>100</THRESHOLD_INTERSECTIONS>
            <PROBABILISTIC>TRUE</PROBABILISTIC>
            <MIN_LIN_LENGTH>100</MIN_LIN_LENGTH>
            <MAX_LINE_GAP>10</MAX_LINE_GAP>
            <ASHOW>TRUE</ASHOW>
            <COLOUR>RED</COLOUR>
            <WINDOWSNAME>HOUGH_RESULT</WINDOWSNAME>
      </FUNCTION>
</MODULE>
```

http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

## 1.4 COMPUTE FEATURES

### 1.4.1 Filter Objects

Given a set of objects eliminates the objects that are out of a selected range of a feature.

| FILTER OBJECTS | Possible Values | Default assignment |
|---|---|---|
| LOAD_OBJECTS | Objects to analyze. | “” |

| | | |
|---|---|---|
| BY | SIZE Objects between minimum and maximum size are kept.<br>DIAMETER Objects between minimum and maximum diameter are kept. | SIZE |
| MINIMUM | Minimum value for the method BY . | 0 |
| MAXIMUM | Maximum value for the method BY. Use 0 for not having higher threshold. | 0 |
| SAVE_OBJECTS | Saved Resultant objects. | “” |

```
<MODULE seq="014" name="COMPUTE FEATURES">
      <FUNCTION name="FILTER OBJECTS">
            <LOAD_OBJECTS>NUC_OBJ</LOAD_OBJECTS>
            <BY> SIZE </BY>
            <MINIMUM>1</MINIMUM>
            <MAXIMUM>30</MAXIMUM>
            <SAVE_OBJECTS>BODIES_FILTERED</SAVE_OBJECTS>
      </FUNCTION>
</MODULE>
```

### 1.4.2 Extracting BASIC features

Statistical features are related to the intensity inside objects and its statistical distribution. Intensity values are related to the amount of markers in a location (amount of intensity is proportional to amount of labeled substance). They can also be used as co-localization patterns if different channels are correlated. The problem with intensity is that it is not a quantitative measure as it can vary between experiments, reagents or staining protocols. Images must be normalized between 0 and 1 before analysis.

| BASIC | Possible Values | Default assignment |
|---|---|---|
| LOAD_OBJECTS | Objects to analyze. | “” |
| REFERENCE | Image in which the objects are referred to. | “” |
| MEAN | If TRUE mean Intensity is calculated. | TRUE |
| SD | If TRUE Standard deviation of intensity is calculated. | TRUE |
| MAD | If TRUE Median Absolute Deviation of intensity values is calculated. | TRUE |
| QUANTILES | If TRUE 0.01, 0.05, 0.5, 0.95 and 0.99 quantiles based on the cumulative distribution of intensity are calculated. | TRUE |

| | | |
|---|---|---|
| SAVE_INDIVIDUAL | If TRUE a separated file with the individual features (the features for each object) are stored. | TRUE |
| FOUTPUT | Name of the file where the features will be extracted. | “” |
| APPEND_ORIGINAL_NAME | If TRUE appends the original name of the image to the output file. | FALSE |
| APPEND_DATE | If TRUE appends the date to the output file. | FALSE |

```
<MODULE seq="009a" name="COMPUTE FEATURES">
            <FUNCTION name="BASIC">
                  <LOAD_OBJECTS>CELLS_OBJ_F2</LOAD_OBJECTS>
                  <REFERENCE>NUCLEI_NORMALIZED</REFERENCE>
                  <MEAN>TRUE</MEAN>
                  <SD>TRUE</SD>
                  <MAD>TRUE</MAD>
                  <QUANTILES>FALSE</QUANTILES>
                  <SAVE_INDIVIDUAL>TRUE</SAVE_INDIVIDUAL>
                  <FOUTPUT>FEAT_CELLS</FOUTPUT>
                  <APPEND_ORIGINAL_NAME>TRUE</APPEND_ORIGINAL_NAME>
                  <APPEND_DATE>TRUE</APPEND_DATE>
            </FUNCTION>
</MODULE>
```

- **Mean** and **standard deviation** of intensity values.
- **MAD**, Median Absolute Deviation from intensity values.
- **Quantile intensity** in a specified region. Can be customized, but common quantiles are 0.01, 0.05, 0.5, 0.95 and 0.99.

### 1.4.3 Extracting SHAPE features

| SHAPE | Possible Values | Default assignment |
|---|---|---|
| LOAD_OBJECTS | Objects to analyze. | “” |
| REFERENCE | Image in which the objects are referred to. | “” |
| AREA | If TRUE size in pixels is calculated. | TRUE |
| PERIMETER | If TRUE perimeter of object in pixels is calculated. | TRUE |
| RADIUS | If TRUE minimum, average and maximum radius in pixels is calculated. | TRUE |

| | | |
|---|---|---|
| ROUNDNESS | If TRUE roundness coefficient is calculated. | TRUE |
| SAVE_INDIVIDUAL | If TRUE a separated file with the individual features (the features for each object) are stored. | TRUE |
| FOUTPUT | Name of the file where the features will be extracted. | "" |
| APPEND_ORIGINAL_NAME | If TRUE appends the original name of the image to the output file. | FALSE |
| APPEND_DATE | If TRUE appends the date to the output file. | FALSE |

- **Area:** Number of pixels inside a target outline borders.
- **Perimeter:** Number of pixels around the border of an object. The 8-pixel convention is used (instead of 4 pixel convention).
- **Mean radius:** Mean internal distance from all possible radiuses taken from the center of the object. Maximum and minimum radiuses are also taken (in pixels).
- **Roundness** is a similar measure based again on the relationship area versus perimeter:

$$\frac{4\pi S}{P^2} = \frac{4\pi R^2}{4\pi^2 r^2}$$

In this case, completely round objects are 1 and the less round they are, the closer are to 0.

```
<MODULE seq="009b" name="COMPUTE FEATURES">
      <FUNCTION name="SHAPE">
                  <LOAD_OBJECTS>CELLS_OBJ_F2</LOAD_OBJECTS>
                  <REFERENCE>NUCLEI_NORMALIZED</REFERENCE>
                  <AREA>TRUE</AREA>
                  <PERIMETER>TRUE</PERIMETER>
                  <RADIUS>TRUE</RADIUS>
                  <ROUNDNESS>TRUE</ROUNDNESS>
                  <SAVE_INDIVIDUAL>TRUE</SAVE_INDIVIDUAL>
                  <FOUTPUT>FEAT_CELLS</FOUTPUT>
                  <APPEND_ORIGINAL_NAME>TRUE</APPEND_ORIGINAL_NAME>
                  <APPEND_DATE>TRUE</APPEND_DATE>
            </FUNCTION>
      </MODULE>
```

### 1.4.4 Extracting MOMENT features

| MOMENT | Possible Values | Default assignment |
|---|---|---|
| LOAD_OBJECTS | Objects to analyze. | "" |

| | | |
|---|---|---|
| REFERENCE | Image in which the objects are referred to. The value NO_REFERENCE can be used in order to compute the moments without using the intensity values in the calculation. | "" |
| CENTROID | If TRUE x,y coordinates of centroid are stored relative to the left bottom corner of the image. | TRUE |
| AXIS | If TRUE major and minor axis of the ellipse containing the object, in pixels, is calculated. | TRUE |
| ECCENTRICITY | If TRUE eccentricity value is calculated. | TRUE |
| THETA | If TRUE theta angle is calculated. | TRUE |
| SAVE_INDIVIDUAL | If TRUE a separated file with the individual features (the features for each object) are stored. | TRUE |
| FOUTPUT | Name of the file where the features will be extracted. | "" |
| APPEND_ORIGINAL_NAME | If TRUE appends the original name of the image to the output file. | FALSE |
| APPEND_DATE | If TRUE appends the date to the output file. | FALSE |

- **Center of mass in x and y coordinates**, relative to the upper left corner of the image (in pixels).
- **Elliptical fit major and minor axis** (in pixels). A line connecting two pixels on the object boundary is called a chord. For many shapes, the length and orientation of the longest possible chord between boundary pixels fits in an ellipse, and it is equivalent to its major axis. It can be used to indicate size and object orientation.
- **Eccentricity:** elliptical eccentricity defined by:

$$\sqrt{\frac{1 - major.axis^2}{minor.axis^2}}$$

  Circle eccentricity is 0 and straight line eccentricity is 1.
- **Theta:** Central moments can be used to calculate the axis of the least moment of inertia. For many objects it corresponds to the major axis of the ellipse. Theta is the angle of this axis with respect to the x positive axis.

```
<MODULE seq="009c" name="COMPUTE FEATURES">
      <FUNCTION name="MOMENT">
            <LOAD_OBJECTS>CELLS_OBJ_F2</LOAD_OBJECTS>
            <REFERENCE>NUCLEI_NORMALIZED</REFERENCE>
            <CENTROID>TRUE</CENTROID>
            <AXIS>TRUE</AXIS>
            <ECCENTRICITY>TRUE</ECCENTRICITY>
            <THETA>TRUE</THETA>
            <SAVE_INDIVIDUAL>TRUE</SAVE_INDIVIDUAL>
```

```
            <FOUTPUT>FEAT_CELLS</FOUTPUT>
            <APPEND_ORIGINAL_NAME>TRUE</APPEND_ORIGINAL_NAME>
            <APPEND_DATE>TRUE</APPEND_DATE>
        </FUNCTION>
    </MODULE>
```

### 1.4.5 Extracting HARALICK features

**Haralick texture features.** A texture is a pattern that is repeated along an object in an image (E.g. rough surface, stripped pattern, dots, etc.). In Haralick´s paper**()**, the authors gathered a set of functions that can be used to classify and discriminate textures. Some of these functions involve statistical approaches of autocorrelation function, gray tone co-occurrence or entropy related functions. The first step is to compute a correlation matrix. This matrix counts how many pixels are similar to each other. Since evaluating all possible combinations of pixels can be extremely costly, the image is usually divided in bins, where the space is split in portions of n pixels. Thus, user can decide how many neighbors are taken into account when the haralick equations are applied. In a second step, patterns in textures can be better detected if the image is scaled (e.g. removing one row every two). The user can tune both parameters, number of bins, and scales for a better texture classification.

| HARALICK | Possible Values | Default assignment |
|---|---|---|
| LOAD_OBJECTS | Objects to analyze. | “” |
| REFERENCE | Image in which the objects are referred to. | “” |
| BINS | Integer.<br>Usually 8 for 8 bits, 16 for 16 bits, and so on. | 8 |
| SCALE | List of comma separated values for scale distance.<br><br>1,2,4 | 1 |
| FOUTPUT | Name of the file where the features will be extracted. | “” |
| APPEND_ORIGINAL_NAME | If TRUE appends the original name of the image to the output file. | FALSE |
| APPEND_DATE | If TRUE appends the date to the output file. | FALSE |

```
<MODULE seq="018" name="COMPUTE FEATURES">
        <FUNCTION name="HARALICK">
            <LOAD_OBJECTS>NUC_OBJ</LOAD_OBJECTS>
            <REFERENCE>IMAGE1</REFERENCE>
            <BINS>8</BINS>
            <SCALES>1,2</SCALES>
            <FOUTPUT>FEAT_HARALICK</FOUTPUT>
            <APPEND_ORIGINAL_NAME>FALSE</APPEND_ORIGINAL_NAME>
            <APPEND_DATE>TRUE</APPEND_DATE>
        </FUNCTION>
```

```
</MODULE>
```

Haralick features are not treated like the previous features. Since they are texture features, there is no need to store the mean and general values. Otherwise, the user can import the data in R or Matlab and do it by himself.

Secondly, the file is stored independently because the concept of bins and scales. Different bins can give different results, and also, the more scales we use, the more data we have.Adding this to the previous features file gives us an enormous file, so at the end is more comfortable having the Haralick features in an independent file.

Total extracted features 13:
ASM, CON, COR, VAR, IDM, SAV, SVA, SEN, ENT, DVA, DEN, IMC1, IMC2

- angular second moment (ASM),
- inverse diff mom (IDM),
- difference Variance (DVA)
- difference entropy (DEN)
- entropy (ENT)
- correlation (COR)
- contrast (CON)
- variance (VAR)
- sum average (SAV)
- sum entropy (SEN)
- sum variance (SVA)

More information in:
R. M. Haralick, K. Shanmugam, and I. Dinstein, “Textural Features for Image Classification,” *IEEE Trans. Syst. Man Cybern.*, vol. 3, no. 6, pp. 610–621, Nov. 1973.

### 1.4.6 All features

It allows you to execute the previous functions (BASIC,SHAPE, MOMENT and HARALICK) in batch.

| ALL FEATURES | Possible Values | Default assignment |
|---|---|---|
| LOAD_OBJECTS | Objects to analyze. | “” |
| REFERENCE | Image in which the objects are referred to. | “” |
| BASIC | If TRUE BASIC features are calculated. | TRUE |
| SHAPE | If TRUE SHAPE features are calculated. | TRUE |
| MOMENT | If TRUE MOMENT features, both, with and without reference are calculated. | TRUE |

| | | |
|---|---|---|
| HARALICK | If TRUE haralick features are calculated with 8 bins and scales 1 and 2. | TRUE |
| SAVE_INDIVIDUAL | If TRUE a separated file with the individual features (the features for each object) are stored. | TRUE |
| FOUTPUT | Name of the file where the features will be extracted. | "" |
| APPEND_ORIGINAL_NAME | If TRUE appends the original name of the image to the output file. | FALSE |

```
<MODULE seq="004" name="COMPUTE FEATURES">
  <FUNCTION name="ALL FEATURES">
              <LOAD_OBJECTS>BODIES_FILTERED</LOAD_OBJECTS>
              <BASIC>TRUE</BASIC>
              <SHAPE>TRUE</SHAPE>
              <MOMENT>TRUE</MOMENT>
              <HARALICK>TRUE</HARALICK>
              <REFERENCE>BODIES</REFERENCE>
              <SAVE_INDIVIDUAL>TRUE</SAVE_INDIVIDUAL>
              <FOUTPUT>TESTALL</FOUTPUT>
              <APPEND_ORIGINAL_NAME>TRUE</APPEND_ORIGINAL_NAME>
              <APPEND_DATE>TRUE</APPEND_DATE>
        </FUNCTION>
  </MODULE>
```

# 6 The interpreter

NOTE: *This part of the program is still under review. If you experience, a problem with the interpreter, please, send us an email.*

Sometimes we want to perform simple operations with the image. Who knows, maybe is too bright and we want to subtract the mean value, inverse the image or if the image is binary, apply a simple mask using an *or* operation.

Let´s take the show function for example. If want to see how the values of our image will look if we remove the mean value after reduce them to the 70% of the original grey value, we can just type:

```
<MODULE seq="003" name="FILE PROCESSING">
          <FUNCTION name="SHOW">
    <INPUT>(0.7*NUCLEI_NORMALIZED).-MEAN(NUCLEI_NORMALIZED)</INPUT>
              <WINDOWSNAME>NUCLEI</WINDOWSNAME>
              <WIDTH>800</WIDTH>
              <HEIGHT>600</HEIGHT>
              <TYPE>WINDOW</TYPE>
        </FUNCTION>
  </MODULE>
```

The interpreter will take your input image and it will subtract the mean value on each iteration.

The interpreter can be used in every image input or output. If you affect an image in the input, it will modify the image for the current operation. However, if you do it with the output, the operation will be stored.

Here there is a list of operators you can use within the interpreter. Unary operators, they will return you a number as result of applying an operation to the matrix.

**Unary operators:**

- **LOG(Image)**
- **MAX (Image)**
- **MIN (Image)**
- **MEAN (Image)**
- **SD(Image)**
- **SQRT(Image)**
- **TRACE(Image)**
- **NOT(Image)**

Binary operators they need one matrix at least. You can operate matrix with matrix or matrix with an scalar number.

**Binary operators:**

- **"+" sum**
- **"-" subtract**
- **".-" subtract (absolute difference, no negative values at the end)**
- **"*" multiply**
- **"/" divide**
- **"&" and**
- **"|" or**
- **".*" Use it only with matrices. Per element matrix multiplication.**
- **"^" exponent. The right side must be a number.**

# ERRORS LIST

**For debugging purposes**

**Error -1907**:”ERROR: Invalid regular expression, not matching files found”.
**Error -1908 :** "ERROR: unequal number of files. Must supply same number of files for every channel"

**Error -1910 :** "Error reading file" in Load Image, after trying to load an image.
**Error -1911 :** "Normalization cannot be done. Only images with 8 and 16 bits allowed."
**Error -1912 :** "Error during writing: File format for saving not recognized."
**Error -1913: “**ERROR: Cropped image (ROI) out of valid boundaries.”
**Error -1922**: ”ERROR: Adaptive threshold in not allowed mode”.
**Error -1945**: ”ERROR: All input images must have the same size.”

**Error -1946**: ”ERROR: Non existent operation.”

**Error -2011** : "XML file not well formed. Error loading XML file."
**Error -2013 :** "ERROR: bad input parameters \n" - Improper number of arguments

**Error -2020** : "ERROR: closing tag."
**Error -2021** : "ERROR: closing tag not found for an element."
**Error -2022** : "ERROR: Root element not found."
**Error -2023** : "ERROR: Node next to root is not a module."
**Error -2024** : "ERROR: Node next to module is not a function."
**Error -2025** : "ERROR: Node next to module is not a function."
**Error -2026** : "ERROR: XML file could not be load."
**Error -2027** : "ERROR: Module does not exist."
**Error -2028** : "ERROR: Function does not exist."
**Error -2029** : "ERROR: Bad Conversion parameter in function.”
**Error -2030** : "ERROR: In parameters inside function.”
**Error -2054** : "INVALID directory" - directory is not valid.