ToolIP
Tool for Image Processing

Version 2024

User Manual

Fraunhofer ITWM

www.itwm.fraunhofer.de/toolip
toolip@itwm.fraunhofer.de

Contact: Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM
Abteilung Bildverarbeitung
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

Web: `www.itwm.fraunhofer.de/toolip`

E-Mail: `toolip@itwm.fraunhofer.de`

Contents

Chapter 1

Introduction

This manual gives an introduction to the software **ToolIP** (Tool for Image Processing). **ToolIP** has been developed by the image processing department at Fraunhofer ITWM in Kaiserslautern, Germany. As graphical development user interface, it comes with an integrated image processing library developed at ITWM. This library contains a large variety of image processing and analysis algorithms that can be arranged in data flow graphs and thus combined to the solutions of complex inspection problems. Together with its library, **ToolIP** allows for solving complex image processing and analysis problems by visual programming. This approach has the advantage of being very intuitive, and so the user can start working productively after a very short training period. The descriptions of such data flow graphs can be saved and then immediately be used with a command line interpreter called **MAOIcmd**. It is also possible to call a graph from inside a C or C++ application. This makes it easy to integrate a graph developed in **ToolIP** in an inspection system or a solution directly in the production line. **ToolIP** and its library are serving as the basis for many image analysis and quality assurance projects in the academic and industrial field.

Are you interested in trying **ToolIP** for free? An installer of the demo version can be found at the homepage
`www.itwm.fraunhofer.de/toolip`.

The demo version underlies some restrictions, which are explained in section 2.3.2. If you are interested in the licensing conditions or if you want to purchase a run-time or developer version of the underlying algorithms, please do not hesitate to contact Fraunhofer ITWM via
`toolip@itwm.fraunhofer.de`.

It is also possible to use the algorithms from the software **MAVI** (Modular Algorithms for Volume Images) for volume image processing and analysis within **ToolIP**. This toolbox is called **MAVIkit** and offers a comprehensive bandwidth of processing and analysis tools especially designed and useful for volume image data together with a geometrical characterization. Its major field of application is data from materials science.

For obtaining more information concerning or for using **ToolIP** with **MAVIkit** please visit
`www.itwm.fraunhofer.de/mavi`.

This manual is organized as follows:
The installation of **ToolIP** is explained in chapter 2. A first overview over the GUI of **ToolIP** is given in chapter 3. This chapter describes the first steps for building an image analysis graph. The menus and functions of the software are explained in detail in chapter 4. Of special importance are the nodes of a graph since they model the basic algorithms. How these nodes are represented, how parameters are modified and which states the nodes can be in is shown in chapter 5. Some further examples in chapter 7 point out features of the tool useful in practice. In chapter 8 the generation of runtime licenses is described. The documentation of the algorithm nodes can be found in chapter 9. The manual is concluded with a short summary in chapter 10.

Appendix A provides a list of useful shortcuts to make every-day work more efficient. Last but not least, credits to the utilized third party libraries can be found in appendix B.

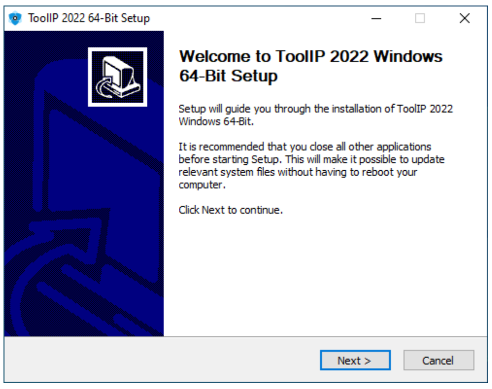
Chapter 2

Installation

In this chapter, we describe the installation procedure of the software **ToolIP**.

2.1 Windows

During the first installation of **ToolIP** administrator rights are required since some environment variables need to be set. If you start the installer suitable for your operating system, you find the setup window as shown in figure 2.1(a). After clicking on Next, you have to agree to the license terms. The corresponding window is shown in figure 2.1(b). If you click I agree, you find the menu as shown in figure 2.2.



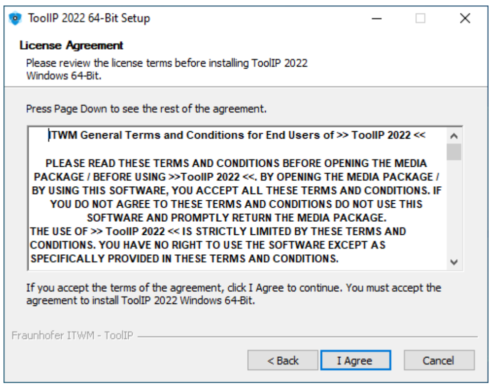
Welcome to ToolIP 2022 Windows 64-Bit Setup

Setup will guide you through the installation of ToolIP 2022 Windows 64-Bit.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

(a)



License Agreement

Please review the license terms before installing ToolIP 2022 Windows 64-Bit.

Press Page Down to see the rest of the agreement.

ITWM General Terms and Conditions for End Users of >> ToolIP 2022 <<

PLEASE READ THESE TERMS AND CONDITIONS BEFORE OPENING THE MEDIA PACKAGE / BEFORE USING >>ToolIP 2022 <<. BY OPENING THE MEDIA PACKAGE / BY USING THIS SOFTWARE, YOU ACCEPT ALL THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS DO NOT USE THIS SOFTWARE AND PROMPTLY RETURN THE MEDIA PACKAGE.
THE USE OF >> ToolIP 2022 << IS STRICTLY LIMITED BY THESE TERMS AND CONDITIONS. YOU HAVE NO RIGHT TO USE THE SOFTWARE EXCEPT AS SPECIFICALLY PROVIDED IN THESE TERMS AND CONDITIONS.

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install ToolIP 2022 Windows 64-Bit.

(b)

Figure 2.1: (a) Start of the installation process. (b) **ToolIP** license agreement.

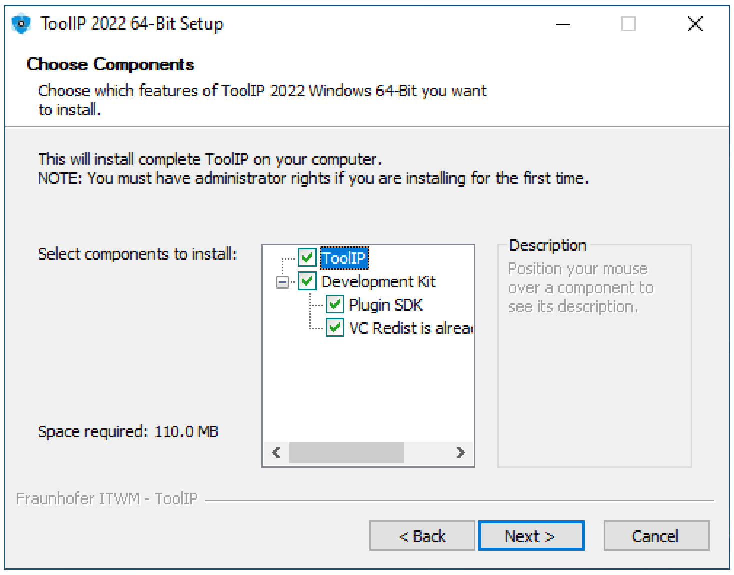
In the drop-down menu, you can choose the option **Basic Install**, which will only install the **ToolIP** package. This might be sufficient if Microsoft Visual Studio 2014 is already installed on your computer. Nevertheless, if the version (including service pack number) of the compiler does not fit the binaries, there will be a problem during execution of the program. In case you are not sure, please tick the box **Vc Redistributable**. Furthermore, a plugin Software Development Kit (SDK) can be installed by ticking the corresponding box. The plugin SDK enables the user to write his own algorithms in C++ and integrate and use them in **ToolIP**.

After clicking on Next, a window appears. This window is shown in figure 2.3. You are asked to give a path for the installation directory. By default this path is set to:

```
C:\Program Files\ToolIP
```

This path is added to a new windows environment variable called `ITWMDIR`.

After you clicked on Next, you are asked to enter your license file name or the directory it is stored in. This is shown in figure 2.4.



ToolIP 2022 64-Bit Setup

Choose Components

Choose which features of ToolIP 2022 Windows 64-Bit you want to install.

This will install complete ToolIP on your computer.
NOTE: You must have administrator rights if you are installing for the first time.

Select components to install:

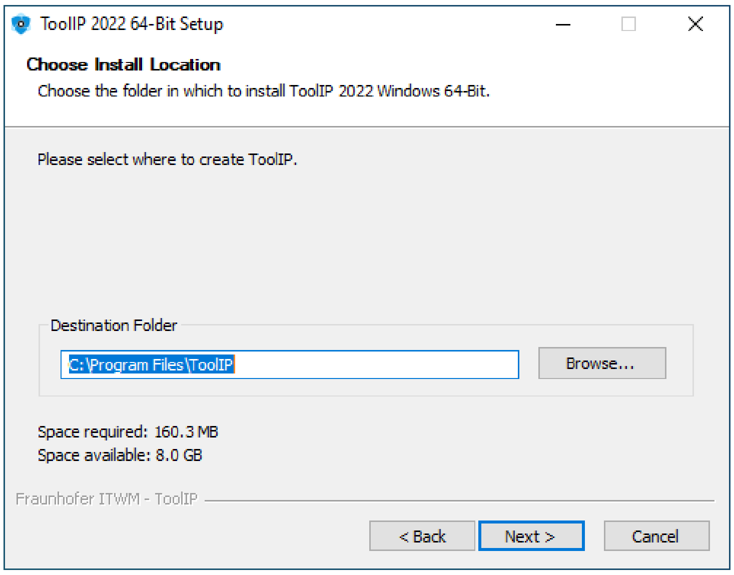
- [✓] ToolIP
- [✓] Development Kit
  - [✓] Plugin SDK
  - [✓] VC Redist is alrea

Description
Position your mouse over a component to see its description.

Space required: 110.0 MB

Fraunhofer ITWM - ToolIP

Figure 2.2: **ToolIP** installer menu.



ToolIP 2022 64-Bit Setup

Choose Install Location

Choose the folder in which to install ToolIP 2022 Windows 64-Bit.

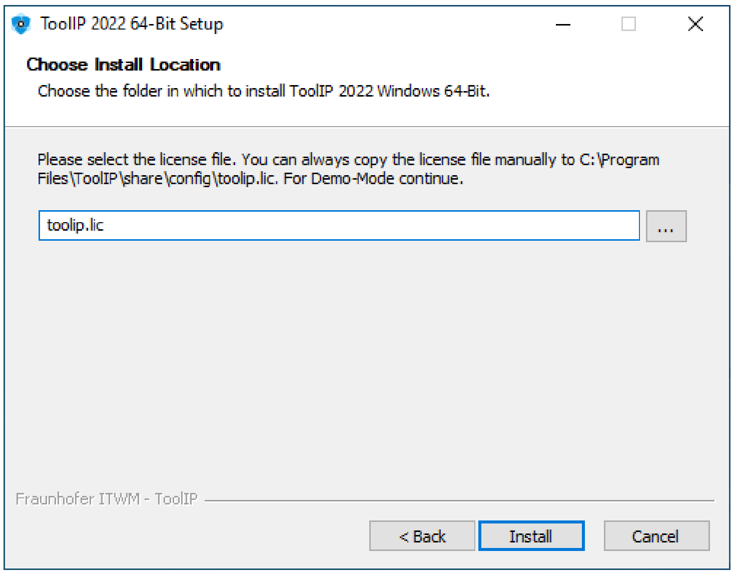
Please select where to create ToolIP.

Destination Folder: C:\Program Files\ToolIP

Space required: 160.3 MB
Space available: 8.0 GB

Fraunhofer ITWM - ToolIP

Figure 2.3: Choose the installation directory.



ToolIP 2022 64-Bit Setup

Choose Install Location

Choose the folder in which to install ToolIP 2022 Windows 64-Bit.

Please select the license file. You can always copy the license file manually to C:\Program Files\ToolIP\share\config\toolip.lic. For Demo-Mode continue.

toolip.lic

Fraunhofer ITWM - ToolIP

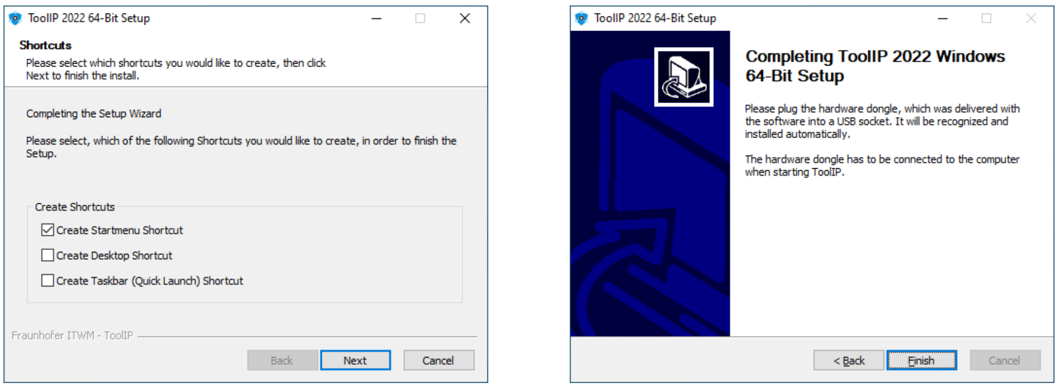
Figure 2.4: Choose the license directory.

By default this path is set to:

```
toolip.lic
```

To choose the right location click onto the three dots next to the path label. After clicking on Install the actual installation procedure starts.

Next, the **ToolIP** installer should show the window in figure 2.5(a). Select the shortcuts you would like to create and click on Next to finish the install. You should now see the window in figure 2.5(b). By clicking the Finish button the installation process is completed and you can start **ToolIP** from the Windows start menu.



ToolIP 2022 64-Bit Setup

Shortcuts

Please select which shortcuts you would like to create, then click Next to finish the install.

Completing the Setup Wizard

Please select, which of the following Shortcuts you would like to create, in order to finish the Setup.

Create Shortcuts

- [✓] Create Startmenu Shortcut
- [ ] Create Desktop Shortcut
- [ ] Create Taskbar (Quick Launch) Shortcut

Fraunhofer ITWM - ToolIP

ToolIP 2022 64-Bit Setup

Completing ToolIP 2022 Windows 64-Bit Setup

Please plug the hardware dongle, which was delivered with the software into a USB socket. It will be recognized and installed automatically.

The hardware dongle has to be connected to the computer when starting ToolIP.

(a) (b)

Figure 2.5: (a) Choose which shortcuts you would like to create. (b) **ToolIP** installer: Installation complete.

2.2 Linux

ToolIP can also be used under some Linux distributions. So far, there is no installer for those platforms available in the download section. If you are interested in using **ToolIP** with Linux, please contact us at
`toolip@itwm.fraunhofer.de`
for further information.

2.3 Starting ToolIP

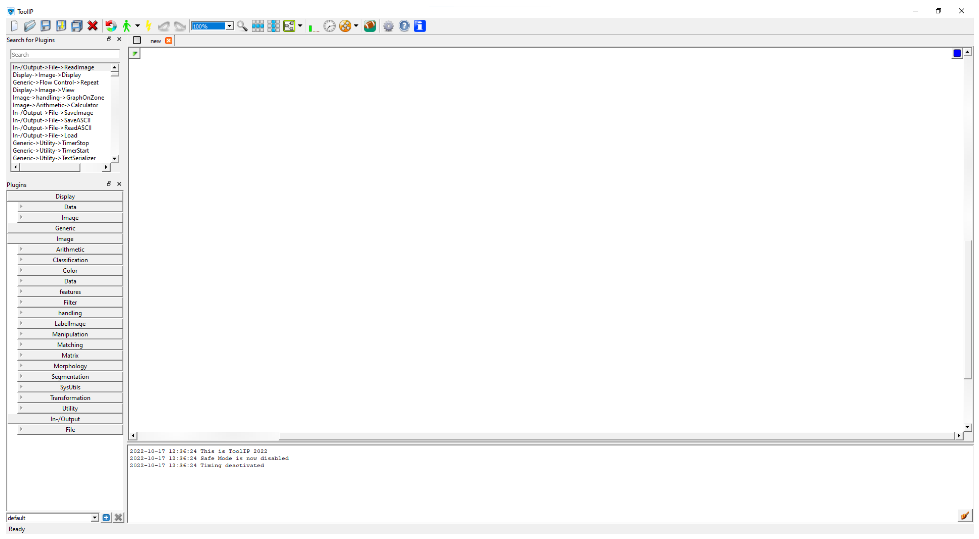
After a successful installation you are now able to start **ToolIP**. Depending on whether you use the full or the demo version, the functionality of your software will be different.

2.3.1 Full Version

The full version of **ToolIP** is delivered with a USB dongle and a license file. The license file must be in the following location for **ToolIP** to find it:

```
%ITWMDIR%\share\config
```

After starting the software with dongle in USB slot the application main window will appear on the desktop. This will look like the screenshot in figure 2.6. You are now able to start using **ToolIP**.



```
2022-10-17 12:36:24 This is ToolIP 2022
2022-10-17 12:36:24 Safe Mode is now disabled
2022-10-17 12:36:24 Timing deactivated
```

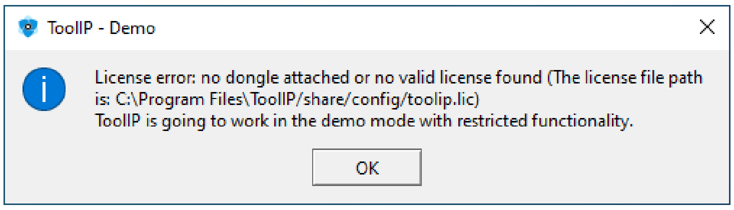
Figure 2.6: The main window of **ToolIP** after the first start.

If the dongle is not found, that is **ToolIP** starts in demo mode even with dongle in USB slot, please re-install the dongle driver. You find the installer in

```
%ITWMDIR%\support\matrix\inf_inst.exe
```

2.3.2 Demo Version

If **ToolIP** is started without dongle or license file, the software will work in demo mode. This is indicated with the message shown in figure 2.7.



License error: no dongle attached or no valid license found (The license file path is: C:\Program Files\ToolIP/share/config/toolip.lic)
ToolIP is going to work in the demo mode with restricted functionality.

Figure 2.7: **ToolIP** will start in demo mode.

Note: The demo mode comes with the following restrictions:

- All input images larger than 256x256x8 pixels are cropped to those respective size in each coordinate direction. This is also the case for **ToolIP**'s command line interpreter MAOIcmd.
- The save option for images is not available.

If you want to purchase a full version, please contact the **ToolIP** mailing address
`toolip@itwm.fraunhofer.de`
for further information.

Chapter 3

Overview

After having finished the installation, we give a short overview over **ToolIP** with very simple image processing examples. A more detailed description of all the functions provided by the program can be found in the following chapters.

All the graphs shown here come with the installation package and can be found in the installation directory, which will be referred to as `INST_DIR`. Please replace `INST_DIR` with your installation directory. To relocate your installation directory, please remember that the environment variable `ITWMDIR` points to it. In the installation directory, the graphs are contained within the subfolder

```
INST_DIR/examples/graphs.
```

You can open the graphs and follow the descriptions given here step by step.

Note:

- If there is no search for plugins window in your **ToolIP** window, you can open it by right-clicking in an empty spot in the menu bar and selecting **Search for Plugins**.
- If there is no list of plugins, you can open it by right-clicking in an empty spot in the menu bar and selecting **Plugins**.
- If you can see the plugin list but it is empty, you have to load libraries by right-clicking in the plugin menu and selecting **Load Library**. For more details, see 4.3.

3.1 First Example – Image Simplification

Let us start with a first example: Figure 3.1 shows **ToolIP** with a minimalistic example graph performing nonlinear diffusion filtering on a gray value image of a bridge. The graph for this example can be found here:

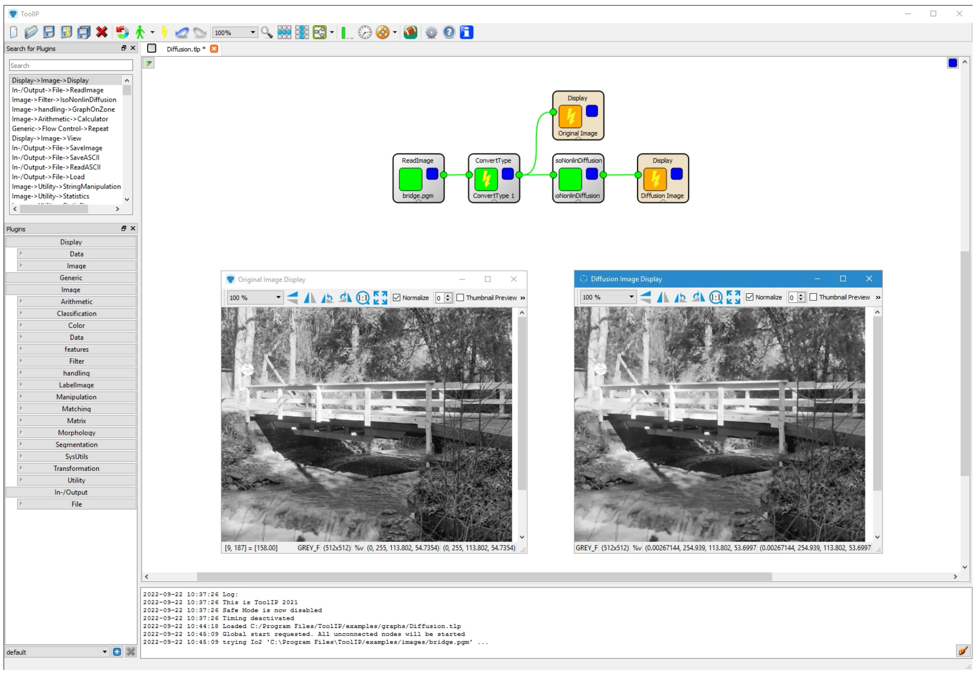
```
INST_DIR/examples/graphs/Diffusion.tlp
```

Please replace `INST_DIR` with your installation directory. During installation, the environment variable `ITWMDIR` has been set to name of the installation directory, too.

You can start the graph in two ways: By clicking on the **Start Graph**-button in the menu bar, all the nodes of the graph are executed, and you can see the results immediately. The nodes can be executed step by step by clicking on the green status field of each node.

In the following, we describe this example in detail: Each of the five boxes in the **ToolIP** workspace – the nodes of the graph – represents an image processing algorithm while the edges between these nodes visualize the data flow.

The graph is shown in figure 3.2. The leftmost node (`ReadImage`) is the source of the data and reads an image from a file. In our example, the image is an 8-bit integer gray value image of size 512x512 pixels. If you are



```
2022-09-22 10:37:26 Log:
2022-09-22 10:37:26 This is ToolIP 2021
2022-09-22 10:37:26 Safe Mode is now disabled
2022-09-22 10:37:26 Timing deactivated
2022-09-22 10:44:18 Loaded C:/Program Files/ToolIP/examples/graphs/Diffusion.tlp
2022-09-22 10:45:09 Global start requested. All unconnected nodes will be started
2022-09-22 10:45:09 trying Io2 'C:\Program Files\ToolIP/examples/images/bridge.pgm' ...
```

Figure 3.1: **ToolIP** with a minimalistic example graph.

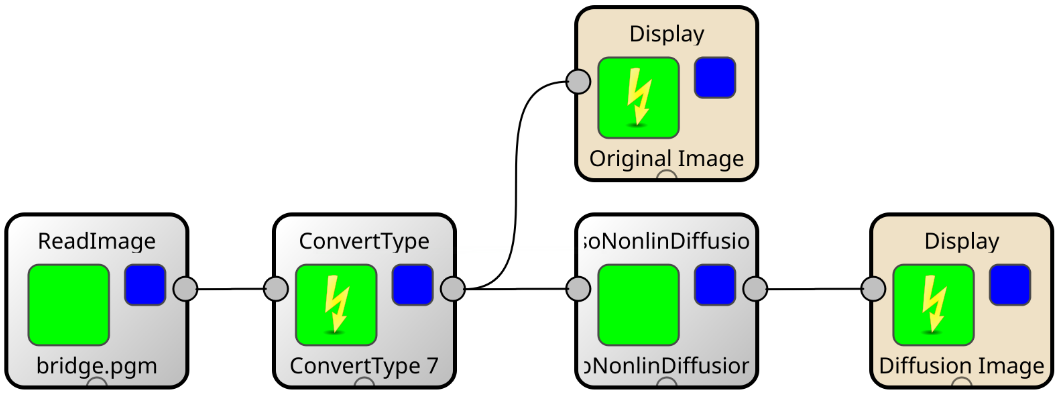


Figure 3.2: Example for a **ToolIP** graph: Simplification of a gray value image with nonlinear diffusion filtering.

running **ToolIP** in demo mode (see 2.3.2), only the upper left corner with size 256x256 is read. The second node (`ConvertType`) converts this image to a floating point data type. This is necessary since the third node (`IsoNonlinDiffusion`) works with floating point precision. This node implements the functionality of an adaptive simplification and denoising of the image data. The remaining two nodes (`Display`) provide the displays of the converted image in floating point precision and the resulting simplified image.

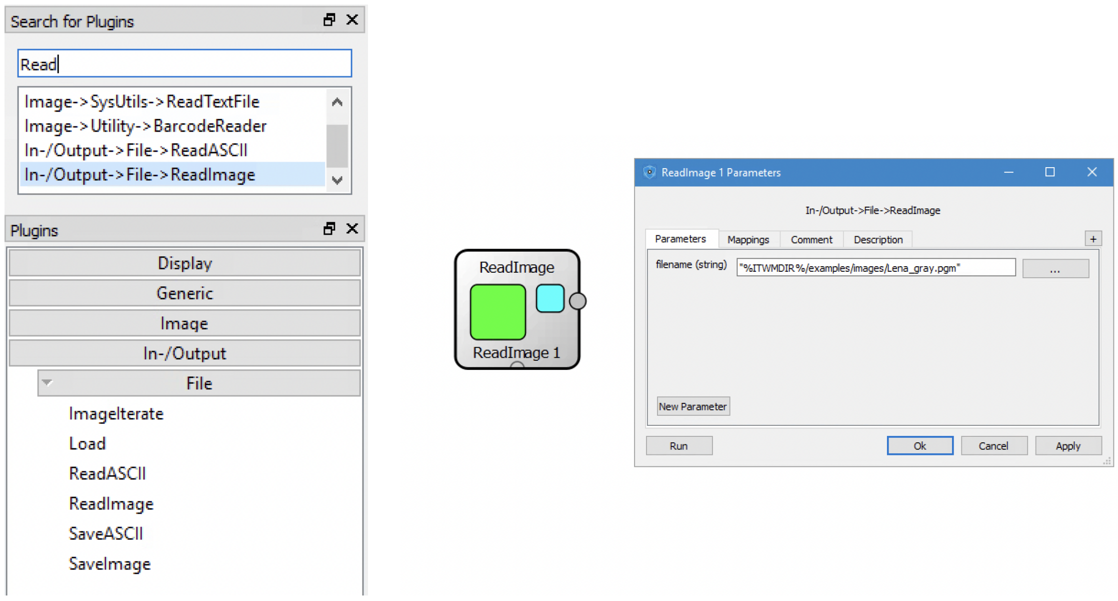
After showing how to execute a graph, we are going to take a look at how to build a new graph in the next section.

3.2 Building Your Own Graph

After starting **ToolIP**, the graph editor starts with an empty workspace (as already seen in figure 2.6). You can select nodes from the plugin menu on the left-hand side and drag-and-drop them into the workspace with the mouse pointer. For more information on the plugin menu, see 4.3. Since the number of nodes can make it complicated to find the desired one fast, the search widget can help to make it easier.

Let us now start with building a very simple example graph with only four nodes. First, we need an input for reading data from file. We propose to use the node **ReadImage** which can be found in the menu under **In-/Output**

→ **File** → **ReadImage**. The node can also be found with the help of the search bar as shown in figure 3.3. Clicking



filename (string) "%ITWMDIR%/examples/images/Lena_gray.pgm"

Figure 3.3: Building your own graph. Left: Find the plugin **ReadImage**. Right: Entering parameters to the **ReadImage** plugin.

on the plugin name either in the search results or in the plugin menu, the plugin can can be drag-and-dropped into the workspace. By clicking on the blue box inside the plugin, the parameter dialog opens. You can now enter the name of the image file to open. Any gray value image can be used for this simple graph, for example

```
%ITWMDIR%/examples/graphs/bridge.pgm
```

Note that the variable name `%ITWMDIR%` is resolved by the application. After typing the parameters and clicking the **OK**-Button, one can run the plugin by clicking on the green box inside the plugin icon. Data available at the

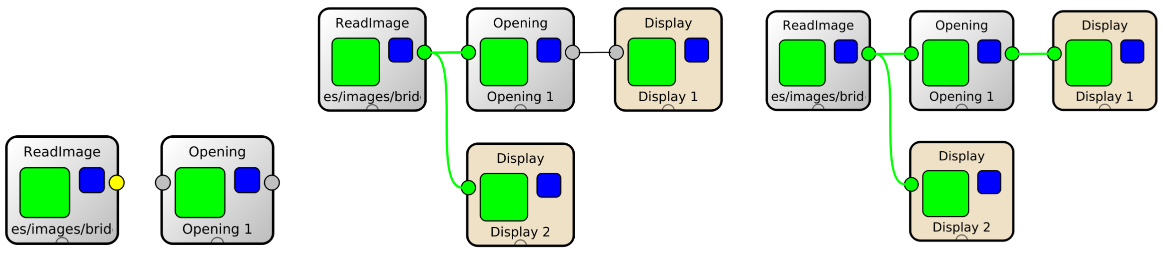
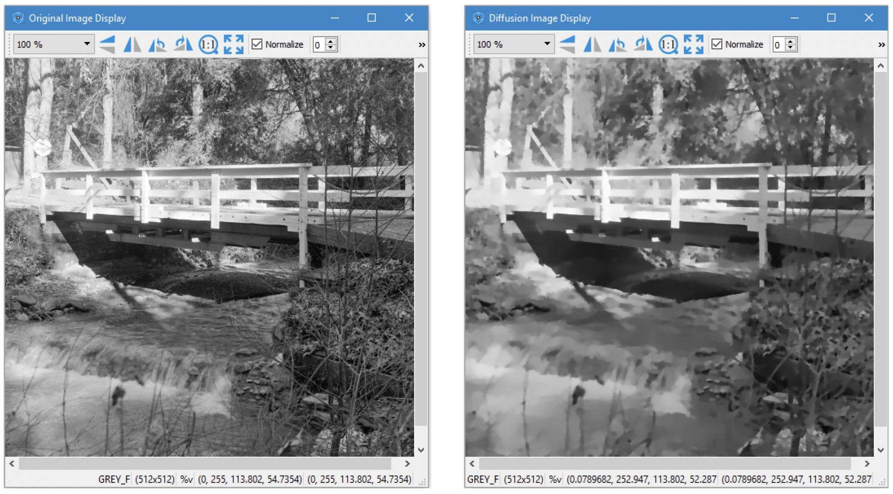


Figure 3.4: Building your own graph. Left: Connecting two plugins. Middle: Green edges indicate existing data. Right: All nodes have run.

output pin of the node is indicated by cyan color. Now you can drag-and-drop the second node for our graph. For this example, we have chosen **Image** → **Morphology** → **Opening** to perform a morphological opening with a rectangular structuring element. To create an edge between the two nodes, first click on the output pin of the **ReadImage** node which then turns to yellow. This can be seen in figure 3.4 on the left. Clicking on the input pin of the **Opening** node creates an edge, i. e. a data connection between these two nodes. We can now add two **Display** plugins to finish the simple graph. The resulting graph is shown in figure 3.4 in the middle. The graph for this example can also be found in the file

```
INST_DIR/examples/graphs/opening.tlp.
```

If you want to execute the graph node by node, it is helpful to see the progress graphically. To this end, green edges indicate that there is data available on this edge, that means the node before this edge has already produced



```
GREY_F (512x512) %v (0, 255, 113.802, 54.7354) (0, 255, 113.802, 54.7354)
```

```
GREY_F (512x512) %v (0.0789682, 252.947, 113.802, 52.287 (0.0789682, 252.947, 113.802, 52.287
```

Figure 3.5: Effect of the morphological opening on the image BRIDGE.PGM.

its output. Black edges indicate that no data is available, and the node after this edge will not be able to run. You can see this in figure 3.4 in the middle and on the right.

The effect of the morphological opening on an example image is displayed in figure 3.5. The rectangular structuring element is clearly visible. Analogously to setting the file name in the **ReadImage** plugin, one can use the parameter dialog of the **Opening** plugin to use a larger structuring element and make the effect of the opening more apparent.

Now you have build your first graph in **ToolIP**. You can save the result by clicking on the floppy disc symbol in the menu bar.

We hope that this chapter could give a first impression of the typical graph development process with **ToolIP**. This first impression will be substanciated by the detailed descriptions of the graphical user interface and the nodes in the following chapters.

3.3 Image data types

With **ToolIP**, most of the data passing between nodes will be represented in the form of images. Depending on the situation, however, images may contain information of different types, e.g. integer-valued or color information. Table 3.1 gives an overview of the different existing image types and their interpretation.

image type	description	range
MONO	just 2 components – foreground and background	0 (background), 1 (foreground)
GRAY8	8 bit integer gray values	$[0,255]$
GRAY16	16 bit integer gray values	$[0,65.535]$
GRAY32	32 bit integer gray values	$[0,4.294.967.295]$
GRAYF	floating point gray values (single IEEE754 precision)	approx. $\left[1.17e^{-38}, 3.40e^{+38}\right]$
COMPLEX	complex floating point gray values (single IEEE754 precision)	approx. $\left[1.17e^{-38}, 3.40e^{+38}\right]^2$ for both real and imaginary components
RGB8	Color image, 3 channels, non-interleaved	$[0,255]$ in every channel
RGB8I	Color image, 3 channels, interleaved	$[0,255]$ in every channel
RGB16	Color image, 3 channels, non-interleaved	$[0,65.535]$ in every channel

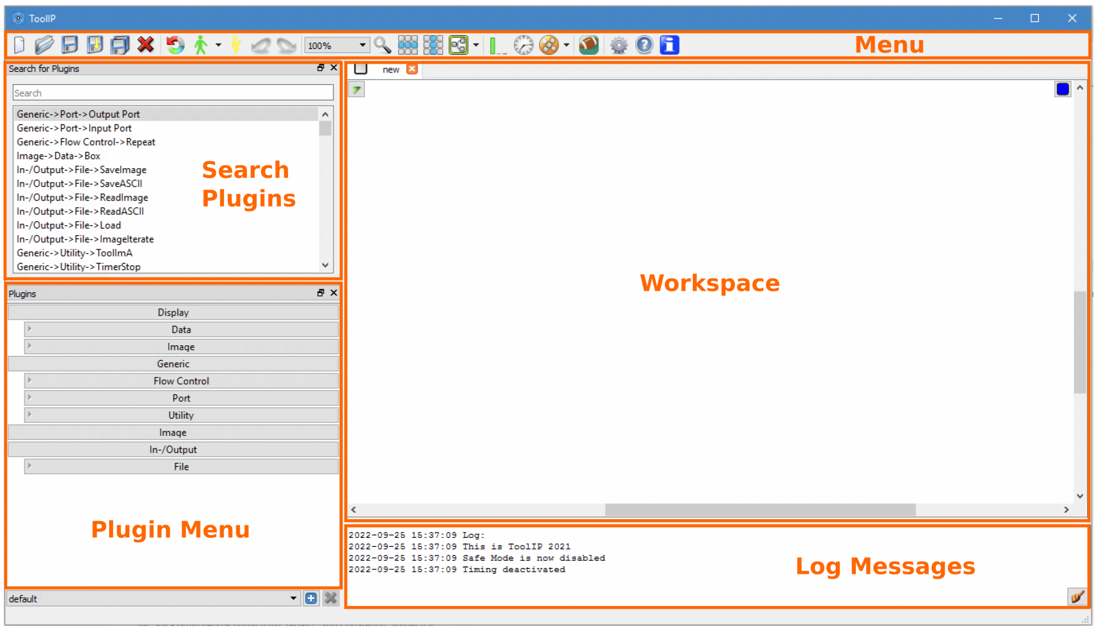
Table 3.1: ToolIP's image types

The most frequently encountered image type within **ToolIP** is GRAYF, i.e., the information stored in every pixel is represented by a floating point number.

Chapter 4

Graphical User Interface

If **ToolIP** is started, the user finds himself in the **main window** as shown in figure 4.1. Herein included are the menu bar, the workspace, the plugin menu, the search plugins and the log window. In the following, these individual elements of the graphical interface and its operation are presented in detail.



```
Generic->Port->Output Port
Generic->Port->Input Port
Generic->Flow Control->Repeat
Image->Data->Box
In-/Output->File->SaveImage
In-/Output->File->SaveASCII
In-/Output->File->ReadImage
In-/Output->File->ReadASCII
In-/Output->File->Load
In-/Output->File->ImageIterate
Generic->Utility->ToolImA
Generic->Utility->TimerStop
```

```
2022-09-25 15:37:09 Log:
2022-09-25 15:37:09 This is ToolIP 2021
2022-09-25 15:37:09 Safe Mode is now disabled
2022-09-25 15:37:09 Timing deactivated
```

Figure 4.1: Main window of **ToolIP**

4.1 Menu Bar

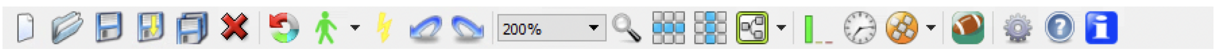


Figure 4.2: Menu bar in **ToolIP**

At the top of the main window is the menu bar (see figure 4.2). On the menu bar almost all important functions of **ToolIP** are available, which are described below.

4.1.1 New

By clicking on the symbol **New** a new tab for displaying graphs opens. A tab is used to create a new graph. If there is an asterisk next to the tab name visible, the file was changed or not yet saved.

Shortcut: `Strg+N`

4.1.2 Open File

Loading an existing graph. By clicking on the icon **Open** the `Open file...` window will be opened for selecting the file to be loaded. The graph will be opened in the current tab in case it is still empty. Otherwise a new tab will be added containing the selected graph.

Shortcut: `Strg+O`

4.1.3 Save

The generated graph will be saved. The first time you save the graph, the window `Save file as...` opens. Here it is possible to specify the saving location. If the graph was already saved before, the current file will be overwritten while saving.

Shortcut: `Strg+S`

4.1.4 Save As

By clicking on the symbol **Save as** the created graph can be saved in a user-specified location.

Shortcut: `Strg+W`

4.1.5 Close Tab

The current tab will be closed. If the graph has been changed, a window appears prompting to save the current graph. Non-saved graphs are indicated by an asterisk symbol in the corresponding tab.

Shortcut: `Strg+Q`

4.1.6 Reset Graph

Resets the current graph to the non-processed state. All outputs of the nodes will be removed.

4.1.7 Start the Graph

Processes all plugins in the graph automatically from beginning to the end. Which node is currently running can be recognised by the status button of the node. Already executed nodes have a green connecting line, those which have not yet been processed, a black one.
When you open the drop-down box of the **Start Graph** button you can select **Infinite execution**, which will perform an infinite execution of the graph. A reset is performed automatically after your graph finished execution.

4.1.8 Stop the Graph

During execution the button turns red. You can always stop the execution process by pressing the **Stop Graph** button.

4.1.9 Execute Consecutively

Enable this feature for single nodes by right-clicking on the green field. This menu bar icon enables the automatic execution. I.e., if an algorithm is started, all consecutive nodes marked with this flash, will be executed automatically, as soon as the input data is available. This mode can be enabled and disabled at any time.

4.1.10 Undo

The symbol **Undo** undoes the last action. (see figure 4.3).

Shortcut: `Strg+Z`

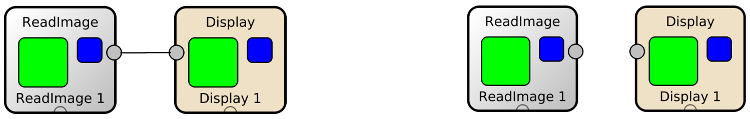


Figure 4.3: Resetting the action **connecting nodes**

4.1.11 Redo

Redo revokes the treatment **Undo**. Has an action been undone, the user can get this action back with **Redo**.

Shortcut: `Strg+R`

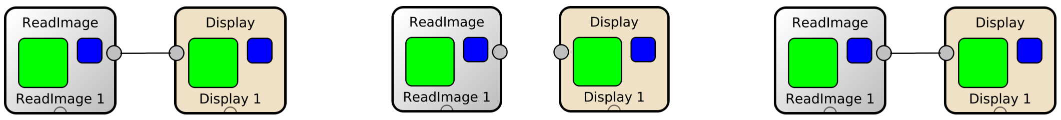


Figure 4.4: Previous undo action withdrawn

Note: The undo button is only available for the following actions:

- adding/deleting a node/an edge
- changing node position (e.g. by dragging)
- port linking/Port reordering
- node parameter change (only if this happens as a result of closing the parameter dialog, changing the parameters during running is not saved)
- graph comment change
- setting flash/stop inside a node

4.1.12 Zoom

The user has the opportunity to set the zoom factor in his workspace. The preset value will be transmitted automatically to all tabs opened. For large graphs it can be useful to choose a lower scale in order to see the complete graph. With the adjustment **fit graph** the zoom settings will be fitted automatically to the current window size. The setting of **fit selection** ensures that all the selected elements are visible.

4.1.13 Search

Search the current graph for nodes. The search can be performed by label, by name or both, it's also possible to search explicitly for defect nodes.

4.1.14 Auto Layout

Arranges all nodes in the current graph automatically, thereby creating an optimal structure for the graph.

4.1.15 Create Subgraph

Moves a selected part of the graph (see Figure 4.5(a)) into a subgraph (see figure 4.5(b)). A marked node can be recognized by a yellow border. By `Ctrl + click` or dragging with the mouse multiple nodes can be selected simultaneously. By clicking on the small turquoise triangle of the subgraph node the subgraph opens in a new window (see figure 4.5(d)).

By clicking on the small green triangle at the left top corner of the open subgraph the window is being attached to the tab bar (see figure 4.5(c)). Subgraphs can be distinguished from main graphs with the help of the corresponding symbol in the tab bar. A main graph is marked with a gray rectangle, subgraphs additionally contain a small dark rectangle in its center. Is a subgraph already open, it will be displayed by a yellow triangle in the subgraph node.

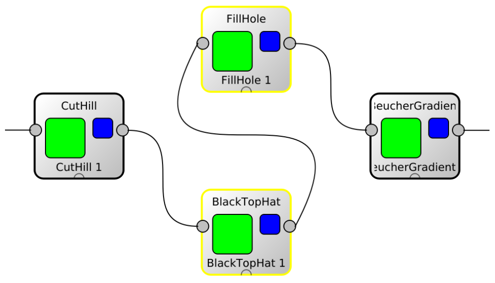
When you open the drop-down menu of the **Create Subgraph** button you can choose *Unfold Subgraph* which will take the contents of the currently selected subgraph and paste it back into the main graph. The connections in the main graph will be updated accordingly. Afterwards, the subgraph will be deleted.

4.1.16 Memory Information

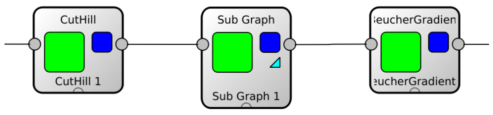
The bar chart provides information about the current memory usage of the graph. A limit can be specified, at what load an automatic deletion of unnecessary data will be performed.

4.1.17 Timing On/Off

Here, a run-time measurement of all nodes in the graph can be activated. By default it is disabled. In active mode, single nodes can be deactivated by selection of the relevant nodes and the use of the access key `Strg+A`. The run-time measurement for each node is shown below the parameter field in milliseconds.
Shortcut: `Strg+T`

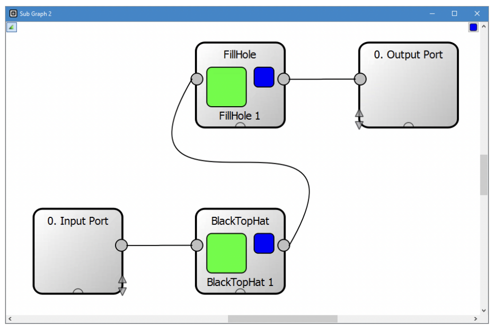


(a) Graph without Subgraph



(b) Graph with Subgraph

(c) Subgraphs tab added



(d) Subgraph representation

Figure 4.5: Create a subgraph in **ToolIP**

4.1.18 Reload Library

For Windows: Pressing the button will open the **Reload Library Dialog**. First, unload all the libraries you want to recompile. After unloading the libraries, you can recompile and install them. Before unloading you cannot install them, since the files are in use by windows (a file which is currently opened cannot be replaced). After recompiling and installing the library you can close the dialog box and all the libraries which were unloaded will be reloaded.

For Linux: This function reloads all the libraries of the currently selected nodes. If no nodes are selected, all nodes in the current workspace are used. The 'Reload library' button has a drop-down menu where you can reload **all** available libraries.

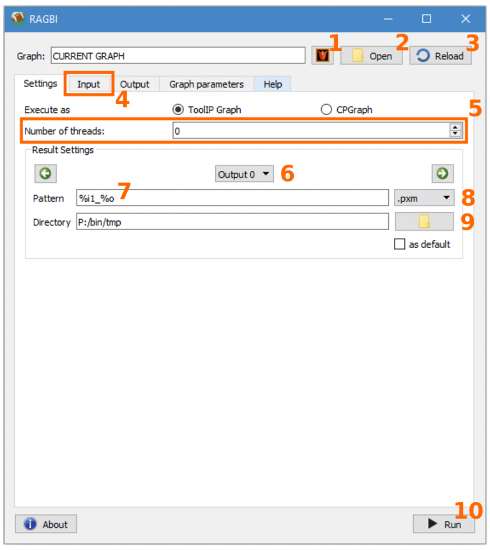
4.1.19 RAGBI - Run A Graph on a Batch of Images or Items

RAGBI is a tool to run the current graph successively on a set of images. A click on the icon opens **RAGBI** in a new window. More details in the next section.

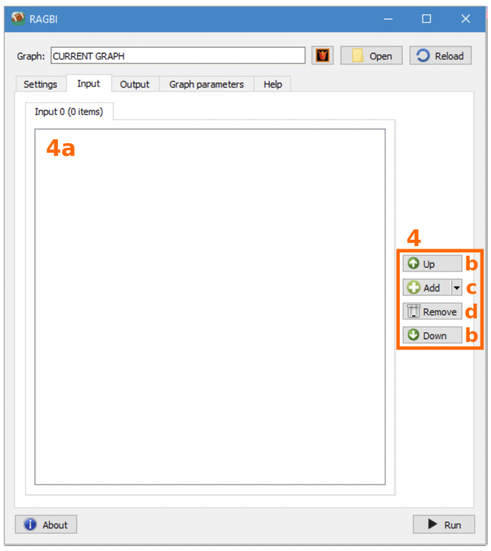
4.2 RAGBI

Figures 4.6(a) and 4.6(b) show the RABGI window. Various things can be customized, most importantly the graph that is used, as well as input and output images. How this is done is now explained in detail:

1. Button to reset the graph that shall be executed to the default, i.e. the currently active **ToolIP** graph ("CURRENT GRAPH")
2. Selection menu to navigate to the graph that shall be executed, default is the graph currently active in **ToolIP**
3. Button to reload the selected graph. This is necessary to include any changes that have been made in the graph since opening the RAGBI window



(a)

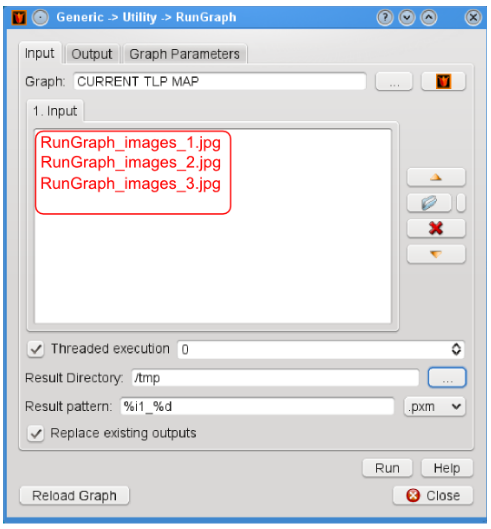


(b)

Figure 4.6: (a) The RunGraph window. (b) The RunGraph input tab.

4. Input image area: Number of input tabs corresponds to the number of input ports of the graph
   (a) List of pictures
   (b) Cursor to move the selected image in the list up/down
   (c) Selection menu to choose input images. This button is inactive if your graph has no input port
   (d) Delete selected images from list
5. Number of threads used to run the graph. Default is to use all available threads (value "0")
6. Selection menu to choose for which output port you want to set (7), (8) and (9)
7. Naming convention for saving the output images of the selected output port. %i[n] stands for the input filenames of the nth input port, i.e. %i1 is replaced by the name of the file in the first input port which contributed to the corresponding output image, %i2 by the name of the file in the second input port and so on. %o is replaced by the output port index (starting from 0)
8. Output image/file format(.pxm/.png/.jpg/.tif/.iass.gz)
9. Path to output directory, i.e. folder where images/outputs are saved
10. Start/Stop graph execution

Note: If your specified output image format (8) is not suitable, RAGBI tries to save the result image in another format, e.g., IMAGE_GREY_F images will be saved in tif-format.



Graph: CURRENT TLP MAP

RunGraph_images_1.jpg
RunGraph_images_2.jpg
RunGraph_images_3.jpg

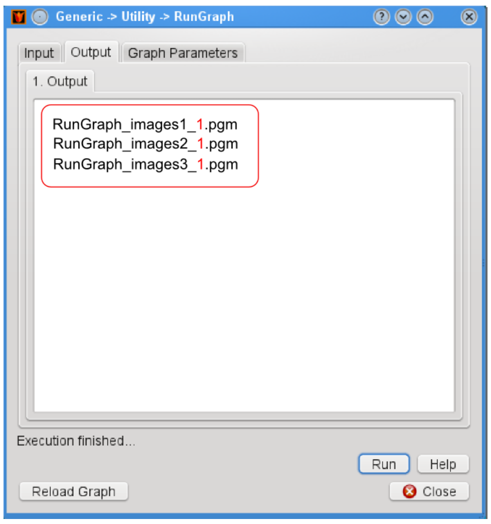
✓ Threaded execution 0

Result Directory: /tmp

Result pattern: %i1_%d .pxm

✓ Replace existing outputs

(a) RunGraph for one input tab

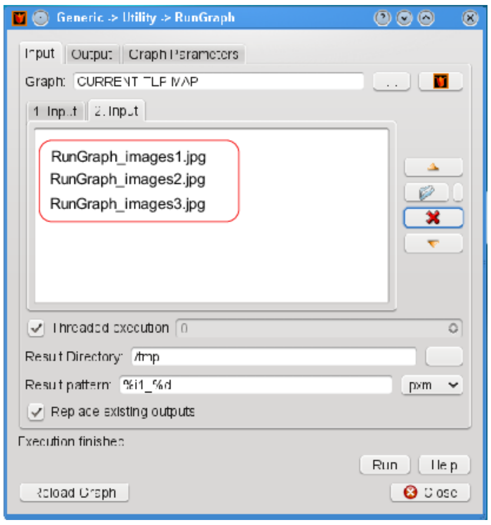


RunGraph_images1_1.pgm
RunGraph_images2_1.pgm
RunGraph_images3_1.pgm

Execution finished...

(b) RunGraph output for one input tab

For further clarification of the output image naming convention: An example, how the output filename will look if there are more than one input ports in the current graph. For the first input the number "1" will be replaced by the first output filename, see figure 4.2(b), if %d appears in the output and the box on (8) is unchecked, no images (from a previous run) will be overwritten. The same will be done for the second input, number "2" will be replaced in the output filename, see figure 4.2(d). This applies for all inputs.



Graph: CURRENT TLP MAP

RunGraph_images1.jpg
RunGraph_images2.jpg
RunGraph_images3.jpg

✓ Threaded execution 0

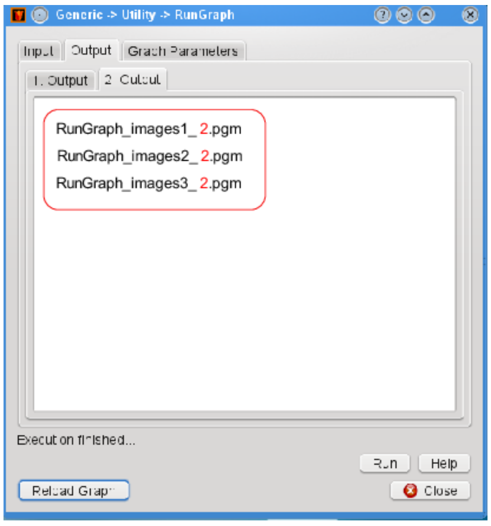
Result Directory: /tmp

Result pattern: %i1_%d pxm

✓ Replace existing outputs

Execution finished

(c) RABGI for two input tabs



RunGraph_images1_2.pgm
RunGraph_images2_2.pgm
RunGraph_images3_2.pgm

Execution finished...

(d) RAGBI output for two input tabs

You can view your input and output images in RAGBI by double-clicking on the item in the list. As you can see in figure 4.7, the path of the displayed image turns bold and italic. If you open an input image, the path of the corresponding output image turns italic and vice versa. With the green arrows above the image, you can navigate through the list of images in the current port.

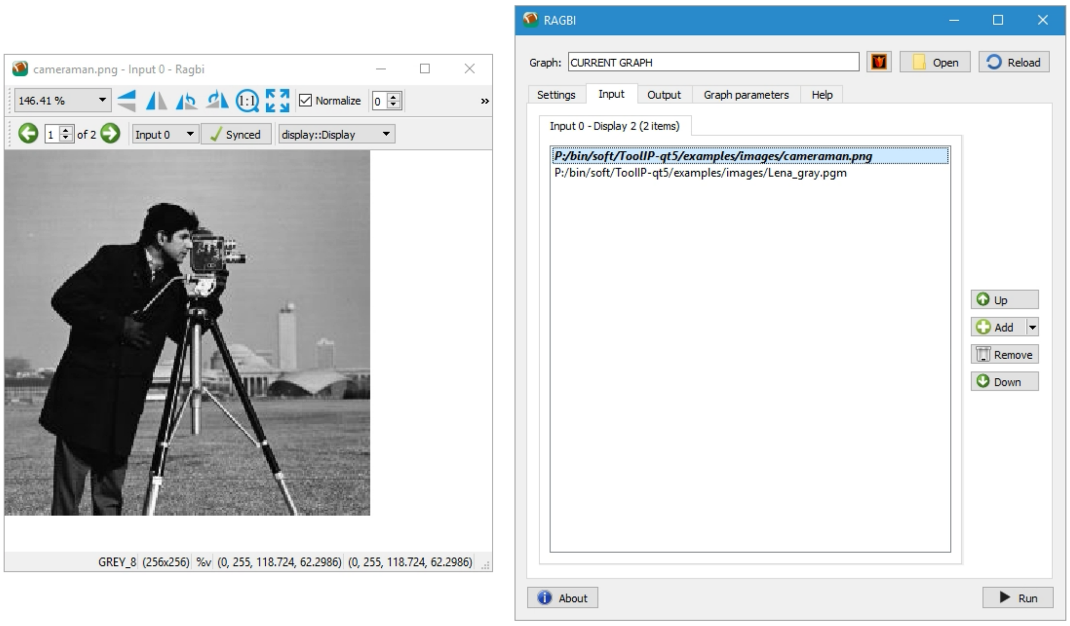

cameraman.png - Input 0 - Ragbi
146.41 %
Normalize
1 of 2
Input 0
Synced
display::Display
GREY_8 (256x256) %v (0, 255, 118.724, 62.2986) (0, 255, 118.724, 62.2986)
RAGBI
Graph: CURRENT GRAPH
Open
Reload
Settings
Input
Output
Graph parameters
Help
Input 0 - Display 2 (2 items)
P:/bin/soft/ToolIP-qt5/examples/images/cameraman.png
P:/bin/soft/ToolIP-qt5/examples/images/Lena_gray.pgm
Up
Add
Remove
Down
About
Run

Figure 4.7: View an input image in RAGBI

You can also open an input and an output image at the same time. If the buttons next to the selected port are on Synced, navigating through the input images (with the green arrows) will simultaneously navigate through the output images and vice versa, see figure 4.8. You can deactivate this function by clicking on Synced, which will then turn to Unsynced.

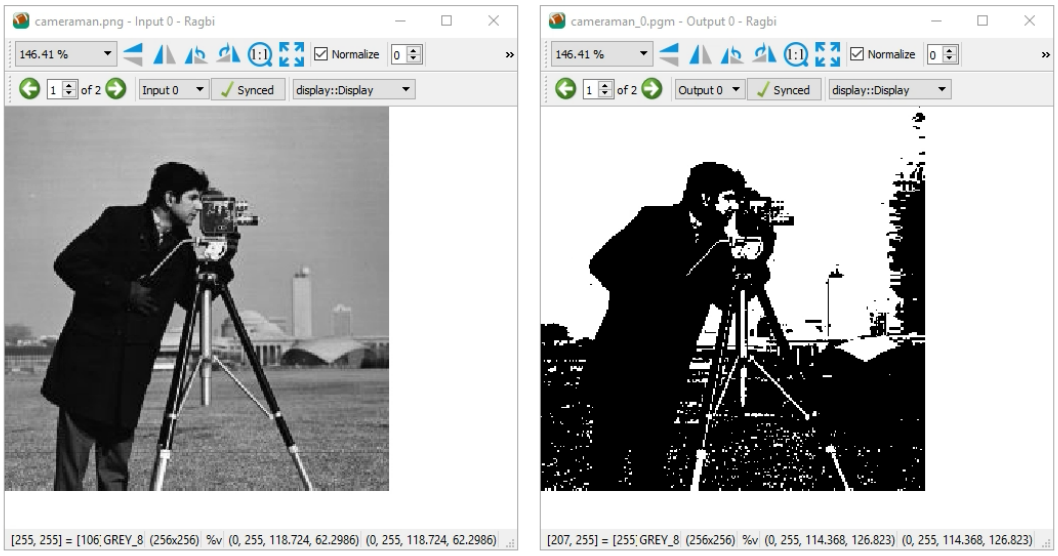
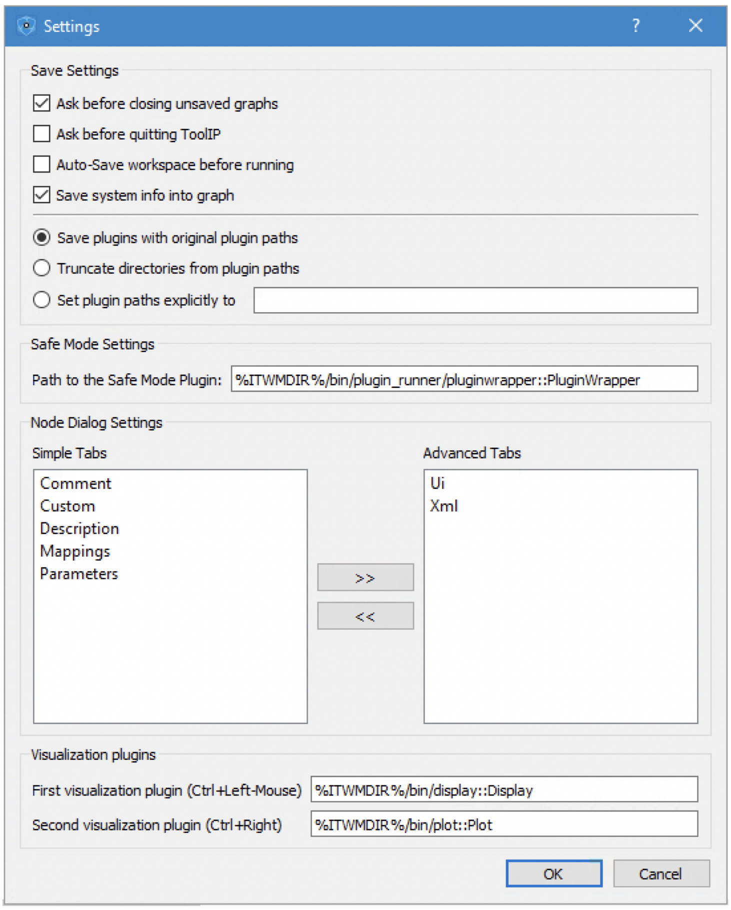

cameraman.png - Input 0 - Ragbi
146.41 %
Normalize
1 of 2
Input 0
Synced
display::Display
[255, 255] = [106 GREY_8 (256x256) %v (0, 255, 118.724, 62.2986) (0, 255, 118.724, 62.2986)
cameraman_0.pgm - Output 0 - Ragbi
146.41 %
Normalize
1 of 2
Output 0
Synced
display::Display
[207, 255] = [255 GREY_8 (256x256) %v (0, 255, 114.368, 126.823) (0, 255, 114.368, 126.823)

Figure 4.8: View input and output image at the same time

4.2.1 Settings

If you click on the gear button, a dialog box will pop-up allowing you to change the global **ToolIP settings**. Here you can set for instance how **ToolIP** saves plugin paths and how the node dialog box is arranged. The settings window is shown in figure 4.9.



Save Settings

- [✓] Ask before closing unsaved graphs
- [ ] Ask before quitting ToolIP
- [ ] Auto-Save workspace before running
- [✓] Save system info into graph

- ● Save plugins with original plugin paths
- ○ Truncate directories from plugin paths
- ○ Set plugin paths explicitly to

Safe Mode Settings

Path to the Safe Mode Plugin: %ITWMDIR%/bin/plugin_runner/pluginwrapper::PluginWrapper

Node Dialog Settings

Simple Tabs	Advanced Tabs
Comment	Ui
Custom	Xml
Description	
Mappings	
Parameters	

Visualization plugins

First visualization plugin (Ctrl+Left-Mouse) %ITWMDIR%/bin/display::Display

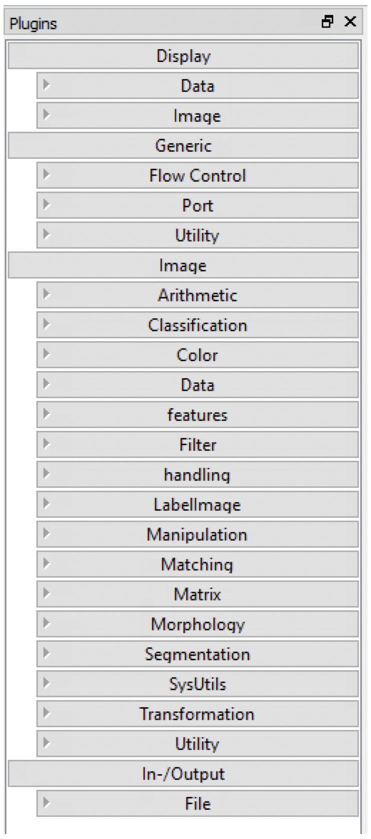
Second visualization plugin (Ctrl+Right) %ITWMDIR%/bin/plot::Plot

Figure 4.9: The settings window

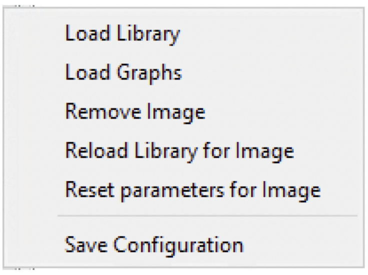
4.2.2 About

Opens an extra window showing information about the running **ToolIP** version.

4.3 Plugin Menu



(a) Plugin menu



(b) Load plugins

Figure 4.10: How to load Plugins in ToolIP

After the first start of **ToolIP** a standard set of plugins is placed in the plugin menu (see figure 4.10(a)). By right-clicking in the plugin menu, there are context menu options **Load Library** and **Load Graphs**, allowing to load additional libraries or graphs (see figure 4.10(b)). In the opening window all plugin libraries of interest can be selected.
As soon as all the additional plugins are loaded they are also displayed in the plugin menu. The currently loaded set of libraries and graphs can be saved by right clicking and choosing **Save Configuration**. Also graphs can be added to the plugin menu. This is especially helpful for the design of re-usable modules. They will also be available in the plugin menu and shown each time **ToolIP** is started.
Removing a plugin is possible by right-clicking on the appropriate plugin and selecting **Remove Item**. It is possible to delete both (sub-)slider and individual plugins. Reloading of libraries is done by right-clicking on a plugin and the selection **Reload Library**.

The user can see and change the properties of a plugin by right-clicking and selection of **Properties**. Another right-click function of the plugin is to reset the parameters of the plugin. If the cursor lingers over a plugin, the **tooltip** will be displayed. Some details about the plugin, the corresponding group, the library name, data input and output as well as parameters are shown.
The plugin menu can be solved, repositioned or closed by the corresponding symbols (at the top right corner) of the **ToolIP** user interface. In the case you want to see the plugin menu again, you can reopen it by right-clicking in an empty spot in the menu bar and selecting **Plugins**. The plugin libraries are sorted by tabs. By selecting the relevant library, the plugin of interest can be moved into the workspace by *Drag & Drop*. Now the plugin can be attached to a graph.

4.4 Search for Plugins

There is a search function for plugins at the top of the plugin menu. The search is equipped with an auto-completion feature. As soon as you type more than two characters in the search box a list of matches for your search query will pop-up. You can *Drag & Drop* the entries from the auto-completion list directly into the workspace. Also adding the selected entry by hitting *Enter*. If you closed the search for plugins, you can reopen it by right-clicking in an empty spot in the menu bar and selecting **Search for Plugins**.

4.5 Workspace

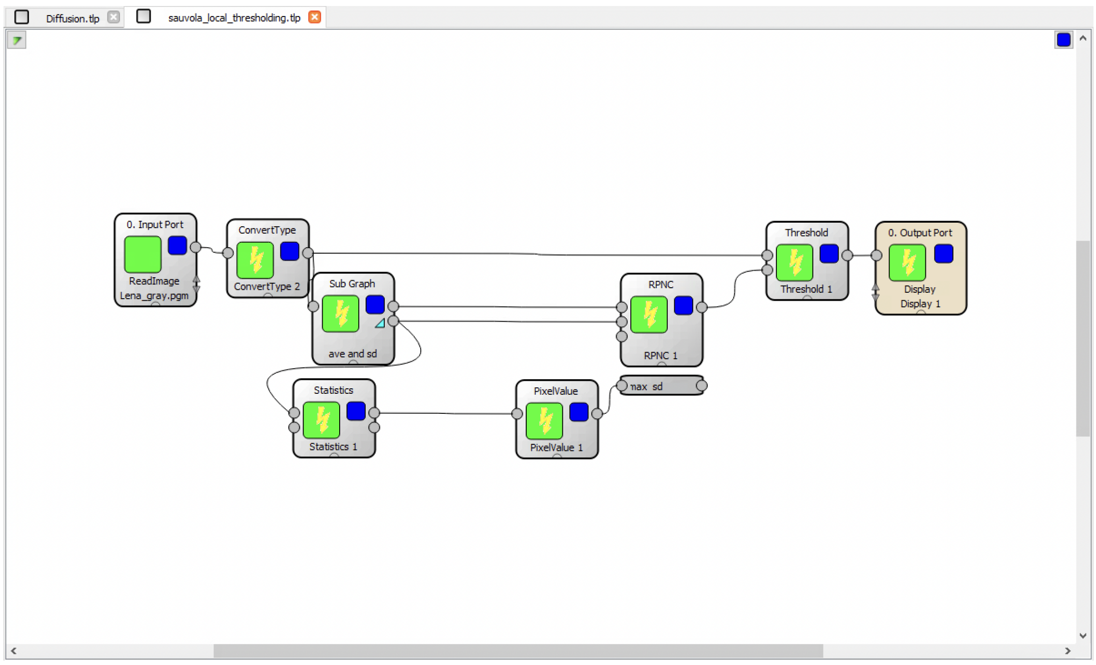


Figure 4.11: **ToolIP**'s workspace

In the workspace the user can include the required nodes and arrange them into graphs (see figure 4.11). At the top of the workspace window, the respective tabs are listed. Each tab contains a graph or a subgraph. The green triangle in the upper left corner allows the user to change the tab into a floating window. By clicking on the triangle the graph will be inserted in the tab-bar again. The save state of the open graph is indicated by a sign in its tab. If an asterisk icon is present, the graph was changed and should be saved before closing the tab. Whether the tab contains a main graph or a subgraph, can be recognized by the corresponding tab icon.

4.5.1 Graph Parameters

By clicking on the blue rectangle in the top right corner the menu box **Graph Parameters** is opened as shown in figure 4.12. Here you are able to define new parameters for your graph and add comments. If you are working on a subgraph, the menu box will be called **Subgraph Parameters** and you can define parameters for your sub graph.

Parameters of nodes inside a subgraph can be defined as parameters of the subgraph. This option is described in detail in section 5.1.2.

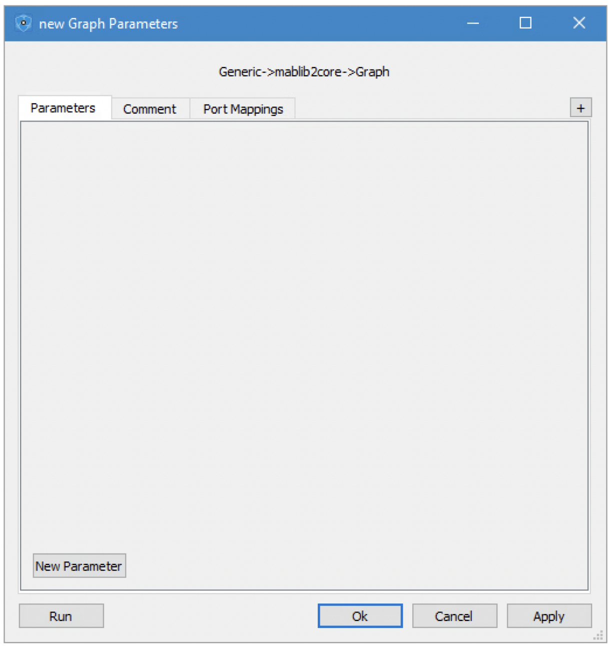
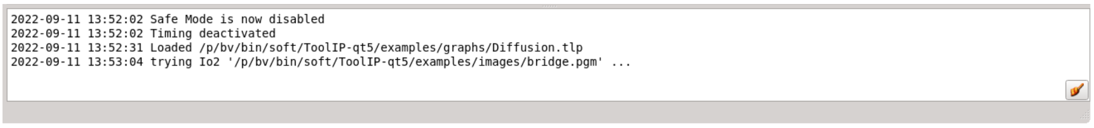


Figure 4.12: Graph Parameters

4.6 Log Window

In the log window, the entire course of the program is recorded. In here error messages, warnings and possible status output of the nodes are displayed. You can clear the messages from the log window by pressing the brush button at the bottom right of the log window.
Additionally to the log window the session is logged to a file in the user's appdata directory.
On Windows, the file can be found in `C:\Users\YOUR_USERNAME\AppData\Roaming\ITWM\ToolIP.log`.
On Linux, the file can be found in `$HOME/.ITWM/ToolIP.log`.



```
2022-09-11 13:52:02 Safe Mode is now disabled
2022-09-11 13:52:02 Timing deactivated
2022-09-11 13:52:31 Loaded /p/bv/bin/soft/ToolIP-qt5/examples/graphs/Diffusion.tlp
2022-09-11 13:53:04 trying Io2 '/p/bv/bin/soft/ToolIP-qt5/examples/images/bridge.pgm' ...
```

Figure 4.13: Log window

Chapter 5

Nodes and Specifications

In the following chapter the different types of nodes and their functionalities are being introduced. In the first section the handling of the basic plugins and the corresponding parameter settings are explained. The latter two sections describe special nodes available for input and output handling of graphs as well as for data flow control.

5.1 Plugins

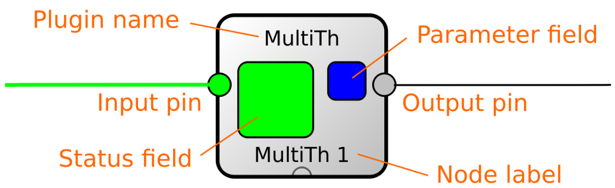


Figure 5.1: ToolIP: Example of a plugin node

A plugin node consists of the elements **Plugin name**, **Status field**, **Parameter field**, **Node label** and data **in- and output pin**. These elements are described in more detail in the following subsections.

5.1.1 Status field

By clicking on the green status field the selected plugin node will be processed. If the node is currently running, the status field turns red and the node label changes into **"running!"**. Since most plugins process very fast this change of label is usually not visible. Processing errors are displayed by a warning. Further details on errors can be found in subsection 5.1.6.

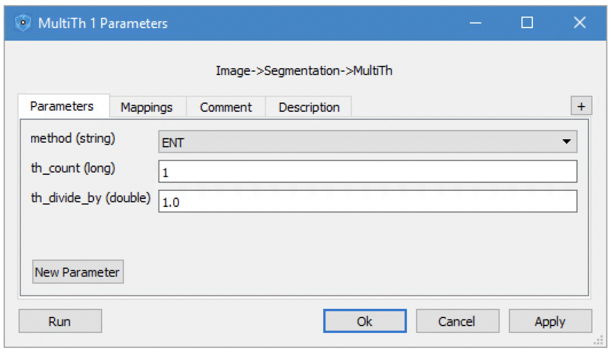
With a single right click on the green status box a bolt appears inside the status field. This status corresponds to a consecutive processing of the plugin. As soon as the running of the previous plugin is finished, the following nodes marked with the lightning symbol are executed automatically. By another two-time right-click the automatic processing function is disabled again. This functionality is also available as the tool bar action **Execute consecutively** and will in this case act as a general circuit for the whole graph.

By two-time right-click on the green status box an executing ban is activated, i.e., further processing of the following branch is prevented. This functionality is useful when during execution certain branches shouldn't be processed. It can be used in combination with the tool bar actions **Start/Stop the graph** and **Execute consecutively** (lightning bolt).

🟧 After execution the color of the status field of the nodes **Display**, **View** and **Plotview** will change to orange. By clicking on it, the display window will disappear.

5.1.2 Parameter Button

🟦 By clicking on the blue parameter button the parameter window (see figure 5.2) will open. In this window all parameters for the plugin can be set. Under the tab **Description** is a brief description of the node.



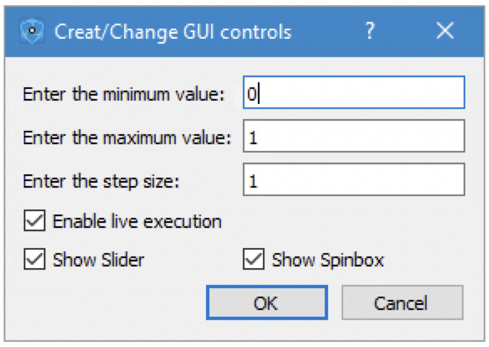
method (string) ENT
th_count (long) 1
th_divide_by (double) 1.0

Figure 5.2: Parameter window of a plugin node

Setting parameters is most comfortable using the tab **Parameters** . Behind each parameter the expected type is shown in parentheses. It is also possible to edit the parameters directly in `XML`-notation by using the tab `XML`. The corresponding `XML`-file of the node will be opened and can be changed. The tab `comments` can be used to add notes to the current plugin. This might for instance be information about specific parameter settings or a brief description of a subgraph.

If the parameter field is opened and you execute the plugin, the plugin is executed with the parameters set in the Node Parameter Dialog. If you click on Cancel inside the dialog box, the parameters are set back to the initial state.

You are able to change a parameter by using a slider. In order to change the display to sliders, you need to right-click inside the setting-field of a parameter and choose the option Create/Change GUI Controls. The window **Create/Change GUI Controls**, shown in figure 5.3, will be opened, where you can set the range of the parameter's slider.

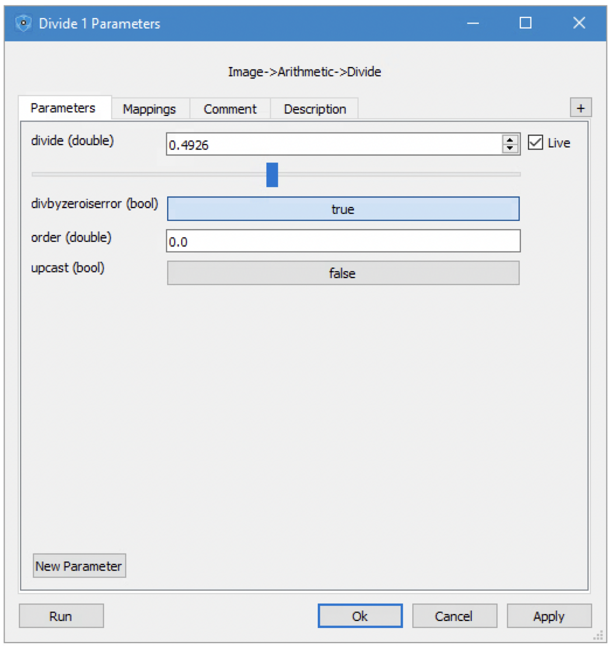


Enter the minimum value: 0
Enter the maximum value: 1
Enter the step size: 1
☑ Enable live execution
☑ Show Slider ☑ Show Spinbox

Figure 5.3: Create/Change GUI Controls

After closing this window, the parameter window will show sliders to enable you to set parameters (see figure 5.4).

It is furthermore possible to add the parameters of a node in a subgraph as a parameter of the subgraph. This can be done in the parameter window of the node at the tab **Mappings**. Here, you need to click on the button **Add Mapping**. In the now opening menu choose the parameter you want to add to your subgraph. The chosen parameter will now appear in the tab **Mapping** next to an input field (see figure 5.5).



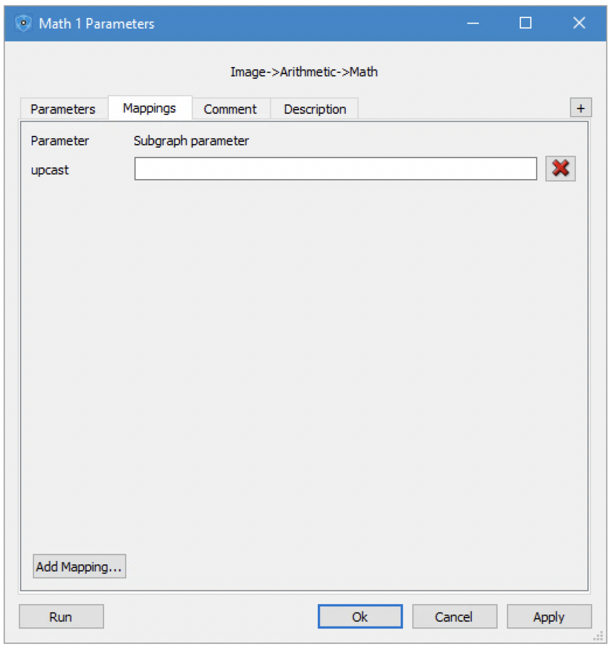
divide (double) 0.4926

divbyzeroiserror (bool) true

order (double) 0.0

upcast (bool) false

Figure 5.4: Parameter window with sliders.

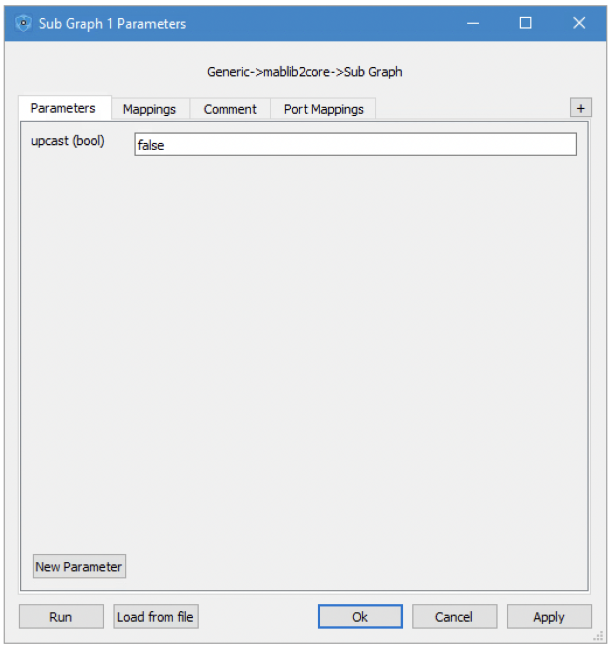


Parameter	Subgraph parameter
upcast	

Figure 5.5: Choose a name for the parameter.

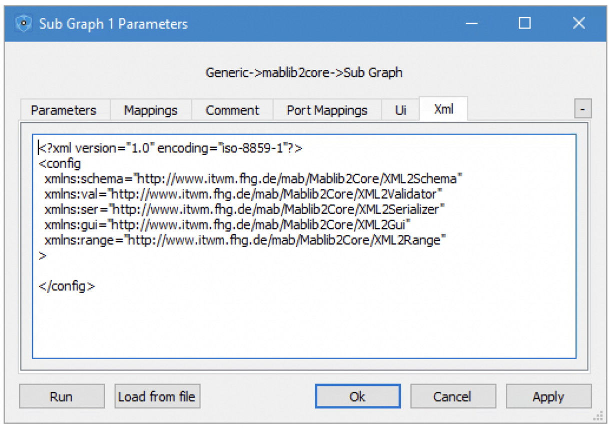
Please define a name for the variable by using this input field. This name will be displayed in the parameter window of the subgraph (see figure 5.6).

You can un-map a node parameter as a subgraph parameter by clicking on the red X next to the respective mapped parameter in the parameter window of the plugin. Graph parameters only make sense if there is at least one plugin inside the graph that actually maps its value to the subgraph. Gray background lineedits at a parameter of a subgraph indicate that there is no plugin inside the graph that maps to this parameter. This is a visual feedback to the user, i.e., this parameter of the graph is no longer necessary or not mapped. At the moment you can only delete this mapping as follows: Choose the tab **Xml**. You have to manually delete the line containing your chosen parameter name until the window looks like in figure 5.7.



upcast (bool) false

Figure 5.6: The node parameter is now a subgraph parameter.



```
<?xml version="1.0" encoding="iso-8859-1"?>
<config
 xmlns:schema="http://www.itwm.fhg.de/mab/Mablib2Core/XML2Schema"
 xmlns:val="http://www.itwm.fhg.de/mab/Mablib2Core/XML2Validator"
 xmlns:ser="http://www.itwm.fhg.de/mab/Mablib2Core/XML2Serializer"
 xmlns:gui="http://www.itwm.fhg.de/mab/Mablib2Core/XML2Gui"
 xmlns:range="http://www.itwm.fhg.de/mab/Mablib2Core/XML2Range"
>

</config>
```

Figure 5.7: This is how the Xml tab should look like after the deletion of all mapped parameters.

If you execute the plugin, while the parameter window is opened, the plugin is executed with the parameters set in the Node Parameter Dialog. If you click on Cancel inside the dialog box, the parameters are set back to the inital state.

Note: Some plugins (matrix::FeatureImage, classification::SVM, manipulation::LabelToFeature,...) use feature plugins as parameters. The current implementation does not work properly when using drag-and-drop from the plugin window. The trick is to specify the feature plugin parameter in advance. This can be done by right clicking on the plugin in the plugin window and selecting the Properties item. Here the feature plugin parameter can be set directly.
In the future, feature plugins will be available as standard plugins and have an online documentation.

Parameter changes can eventually be tested over the button **Run**. Are for all subsequent nodes up to a node of the category *display–>image* the status fields set to Execute consecutively, the changes can be visualized directly. With **OK**, the entries are confirmed. **Cancel** reverses all the changes.

5.1.3 In- and Output Pins

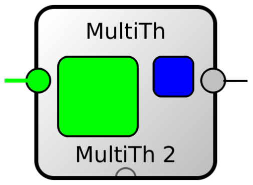


Figure 5.8: Node in- and output pins

Depending on the functionality each plugin contains one or more in- and output pins: The **input pins** are on the left, the **output pins** on the right side of the node. Each output pin can be connected to one or more than one input pins. To each input pin only one output pin can be connected. The connections can be created by selecting and holding a pin and pulling the link to a second pin. By releasing the mouse button the connection is established.
Links can also be generated with a simple click on the two relevant pins. Non-connected pins are shown in gray. If a pin is selected, it will be displayed in yellow. As soon as two pins are connected they will become red. In this state the color of the connector is black, indicating that no processing has been performed, yet. After execution of the first plugin the color of the output pins and the related connectors and input pins become green.

5.1.4 Parameter In- and Output

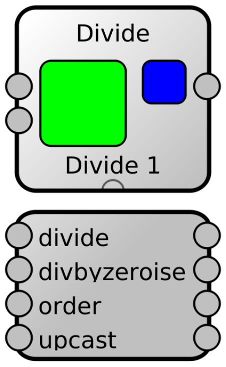


Figure 5.9: Parameter In- and Output

It is possible to define the value of a parameter as the result of another node. When lingering the cursor over the bottom of a node a small gray triangle appears. After clicking on this triangle, the parameter input menu pops out as shown in figure 5.9. Here an output and an input pin is defined for every parameter. It is therefore possible to connect the output pin of another node with the input pin of a parameter. The parameter will then be set to the resulting value of the previous node. An example for this use of the parameter pins is shown in figure 5.10.

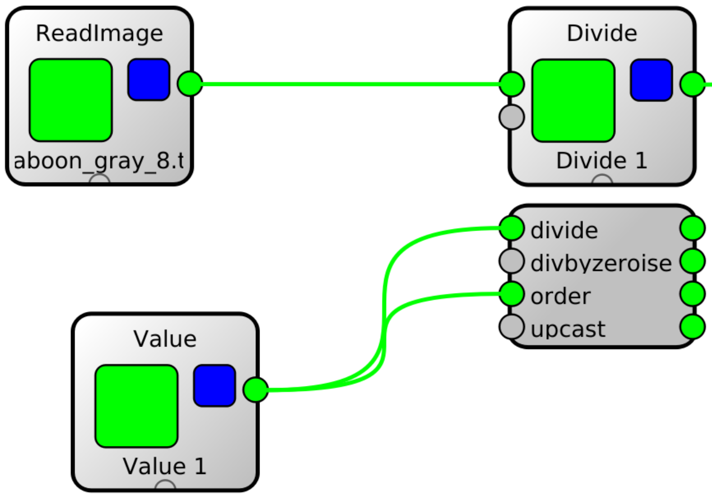


Figure 5.10: Example graph for the use of the parameter inputs

Parameters of subgraphs can also receive input or return output. By clicking on the gray triangle on the bottom of the node of a subgraph, a parameter input menu pops out, which is similar to the one of plugins. Since the parameters of nodes inside a plugin can be added as subgraph parameters as shown in section 5.1.2, the parameter in- and output of subgraphs allows you to define parameters inside a subgraph to be dependent on a node outside the subgraph. Figure 5.11 shows an example, in which the value of a parameter of a Node inside the subgraph is the output of a node in the main graph.

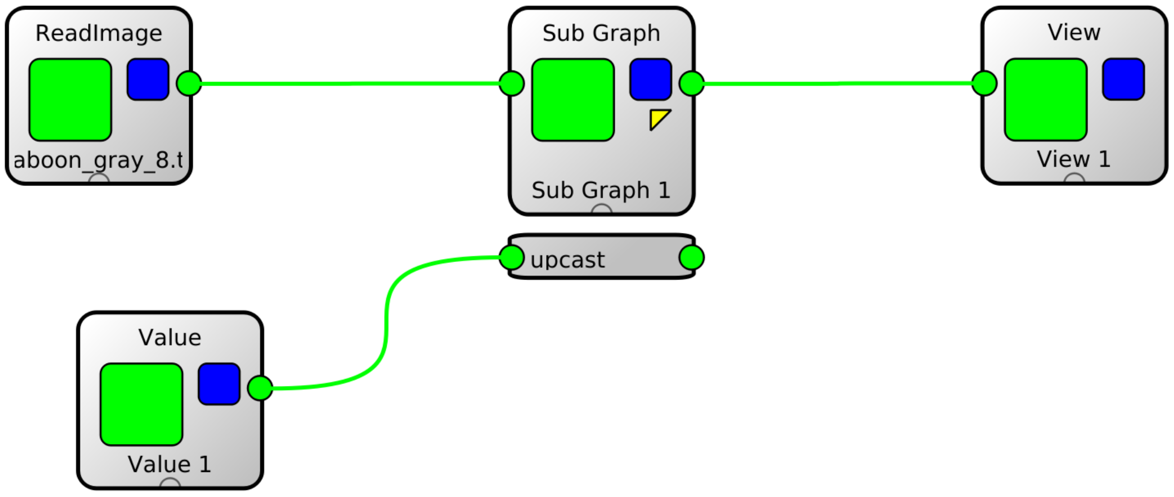


Figure 5.11: Output of node **Value** determines subgraph parameter **upcast**

You can even use the parameter output of another subgraph to be the parameter input of your subgraph (see figure 5.12)

5.1.5 Node Label

By double clicking on the **node label** (below the status field) a window opens to set a new label for this node. The text before the double lines indicates the label of the node. In addition, a text can be written behind these double lines. This text will occur as tooltip by longer linger with the mouse over the node.

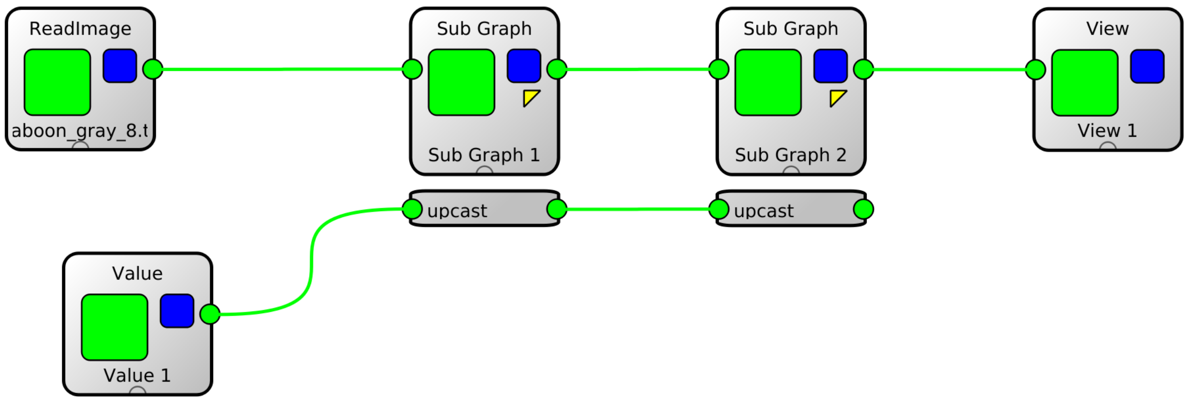


Figure 5.12: Output parameter **upcast** of subgraph 1 is used as input for parameter **upcast** of subgraph 2

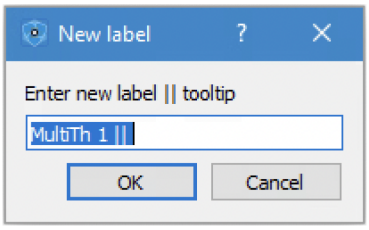


Figure 5.13: Nodes: Changing name and tooltip

5.1.5.1 Automatic Node Labeling

Nodes will always be labeled automatically by the scheme "NAME NUMBER", where NAME corresponds to the plugin name, and NUMBER is a continuous number, which starts at 1 for each **ToolIP** session. If you reload a saved graph, it is not guaranteed, that all nodes will get the same label again, since the labeling is dynamical. But if you give the node a new label, the automatic labeling is over-ruled, and the node will get the same label again.

Note: Plugins like Load or ReadImage inform the user, which image is currently loaded. Therefore this information is directly being displayed by the automatic node labeling. This automatic node labeling is active as long as no user-defined label is defined.

5.1.6 Error Handling

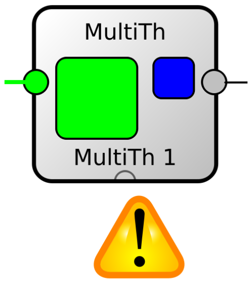


Figure 5.14: Nodes: error handling

If a node could not be executed, a **warning** appears below the node. It is a yellow triangle containing an exclamation mark (see figure 5.14). This might for instance be the case when there is a type mismatch between the data at the input pin and the input data settings of the plugin. By clicking on the warning the error message will be displayed, and the warning will disappear.

5.2 Linking Ports

In **ToolIP** two nodes are available which have a special task in transporting data into and out of a (sub-)graph and have no other functionality. These two nodes, called **Input Port** and **Output Port** can be found under *Generic–>Port*. An example is shown in figure 5.15.

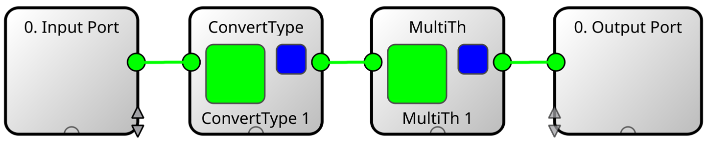


Figure 5.15: In- and output ports

By drag & drop compatible plugins can be attached on top of (be merged into) a port. This way the port acquires the functionality of the plugin. Compatible plugins for input ports are *Load, ReadImage and ReadASCII*, whereas for output ports the compatible plugins are *Display, View, SaveImage, SaveASCII and ScatterPlot*. Plugins can only be attached to the ports on the top level graphs, for subgraphs the attachment functionality is always disabled. Ports do not have a label by default, but they will get one as soon as a compatible node has been added.

Note: In the demo mode the plugins *SaveImage and SaveASCII* are not available and can therefore not be attached to an output port.

5.3 Data Flow Control

In **ToolIP** four nodes are implemented which allow you to control data flow within a graph: **Repeat**, **Branch**, **Merge** and **Switch**. These flow control nodes can be found under *Generic–>Flow Control*.

5.3.1 Repeat Node

The Repeat Node enables the user to include a loop in an algorithm subgraph that is the equivalent to a *for loop* in programming languages. It has one input. The complete subgraph containing the repeat node will be looped.

As with any loop in programming, the user has to specify an increment in the subgraph and provide a loop condition when the repeat node should stop iterating. At the parameter field of the repeat node, under the tab Port Mappings, the user has to specify which output should be mapped onto which input for the next iteration.

To understand how the Repeat Node is used it is recommended to open the example graph

```
INST_DIR/examples/graphs/RepeatRotateLena.tlp,
```

the usage is also shown with additional hints at the key points in figure 5.17.

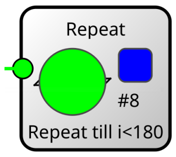


Figure 5.16: Data flow: Repeat

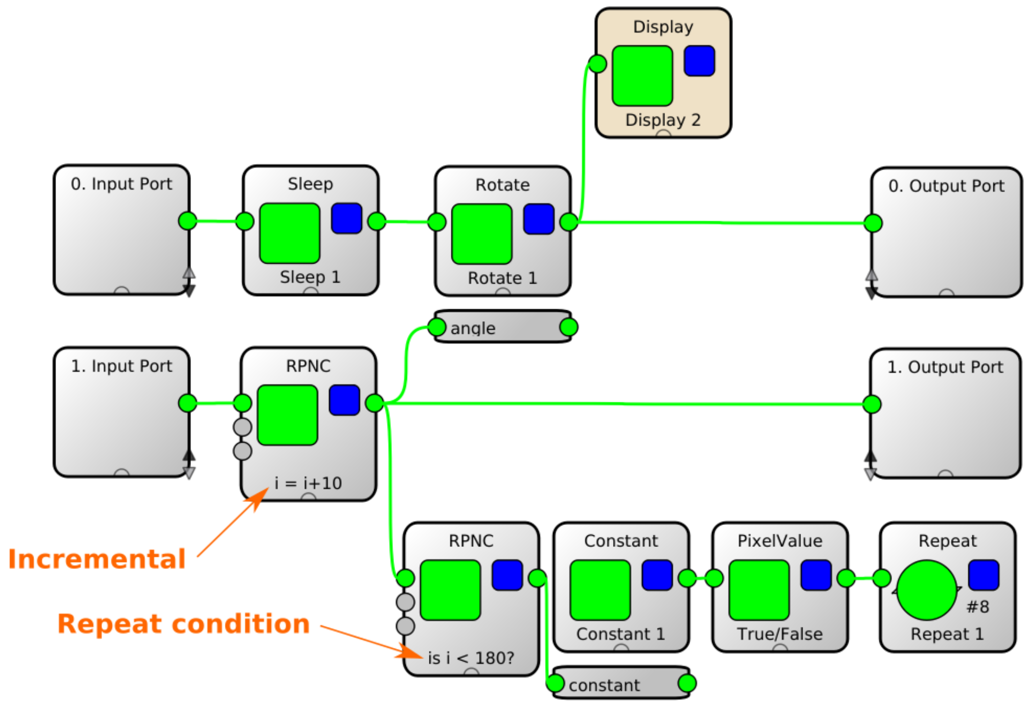


Figure 5.17: Subgraph containing the **Repeat** node.

5.3.2 Branch Node

The branch node allows you to branch out your data flow given some boolean condition. It has two inputs, one for the incoming data (i.e. the image) and one for the branch condition. Depending on the condition it will either place the incoming data onto the first or the second output pin. Thus, the branch node can be thought of as an **if/else-statement**.

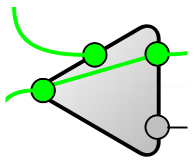


Figure 5.18: Data flow: Branch

The bottom left input is the data input whereas the top left input represents the conditional input. The conditional input has to be of type *CValueBool*. If the condition evaluates to true, the input data will be placed on the first output, otherwise the data will be placed on the second output. If the conditional input pin is not connected to any other node, the data will be placed on the first output.

Figure 5.19 shows how the branch node can be used in practice. An input image is being read using ReadImage. If the value of the input parameter is not equal to zero, a gaussian smoothing is being performed on the image, otherwise the image remains unchanged.

5.3.3 Merge Node

The merge node is the counterpart to the branch node. It allows you to migrate two data streams into one. The merge node forwards its input data to its single output immediately when it arrives. That is, if both inputs of the merge node are connected and if the first input receives data first, it will immediately place this data on its output triggering the next node. Once the second input receives its data it will again trigger the next node with the newly arrived data. The merge node does **not** wait until all input data is available before it executes itself. If being used in combination with the branch node either the first or the second output of the branch node will be further processed.

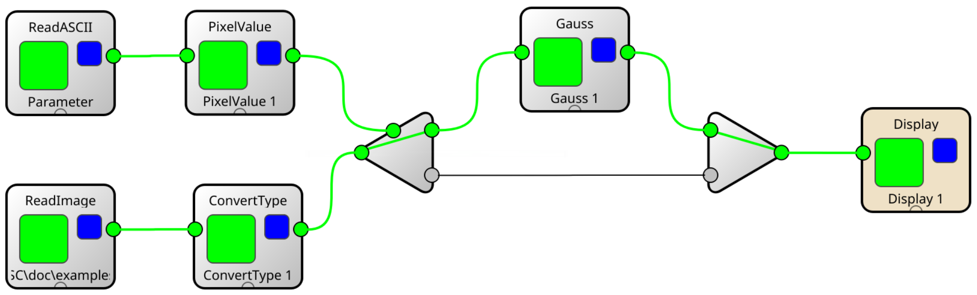


Figure 5.19: Graph using the data flow controls **Branch** and **Merge**

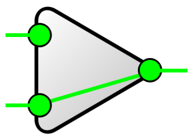


Figure 5.20: Data flow: **Merge**

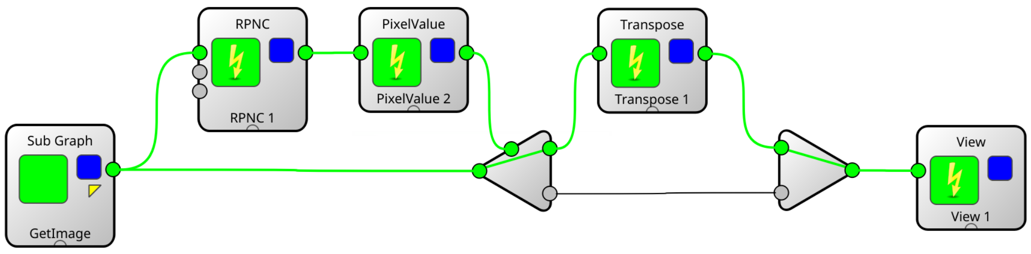


Figure 5.21: Graph using the data flow controls **Branch** and **Merge**

Another example showing the combined use of branch and merge can be seen in figure 5.21. For the input image is being checked whether the image width is larger than its height. If this is the case, the image is being transposed, otherwise the image remains unchanged. The merge node combines the two possible outputs and creates a single output stream.

5.3.4 Switch Node

The switch node is somewhat similar to the branch node. Given some conditional input, it will either forward all of its incoming data to corresponding output or block all execution of following nodes completely. Thus it can be considered as a switch allowing you to turn data flow on or off. The conditional input is the input at the top of the switch node. Again, only *CValueBool* type conditions are allowed. In default mode, every input is forwarded to its corresponding output if the condition is true. However, if you click the *true/false* button once (so that it displays false) it will forward the input data if the condition is false. If no node is connected to the condition pin it will be considered to be true, thus forwarding data if the *true/false button* is set to true, and blocking further execution if it is set to false. By default the switch node comes with only one pair of input/output pins. If additional input/output pairs are needed, they can be added by pressing the + button. If you press this button while holding the *Shift* key, you can remove a pair of input/output pins again.

An alternative way to construct a data flow as shown in 5.19 is by using two switch nodes as shown in 5.23. The top flow puts the input image on the merge node in case the boolean is false. Otherwise, the data stream at the bottom applies a gaussian smoothing on the input image. Only one of the streams is being transferred to the display plugin.

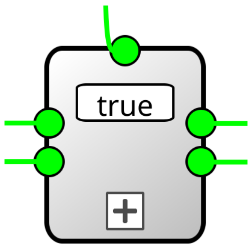


Figure 5.22: Data flow: **Switch**

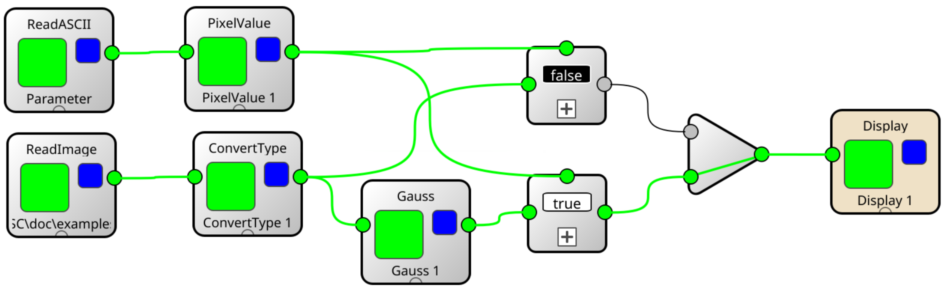


Figure 5.23: Graph using the data flow control **Switch**

5.3.5 Clamp Nodes

A clamp node is a node which only purpose is to create a visually more tidy and appealing graph (cf. Fig. 5.24 and Fig.5.25). It does not process the data in any way but gives you an extra handle on the connections between nodes. You can create a clamp node by double-clicking on a connection and delete it with the *key "del"* or *right-click → Delete*, just as any other node. Additionally, the clamp node is deleted automatically, when the connection(s) it belongs to is deleted. To understand how the clamp node is used it is recommended to open the example graph

```
INST_DIR/examples/graphs/clamp.tlp,
```

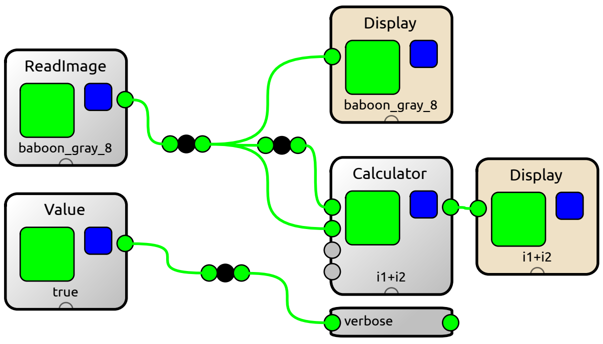


Figure 5.24: ToolIP: Example of a clamp node

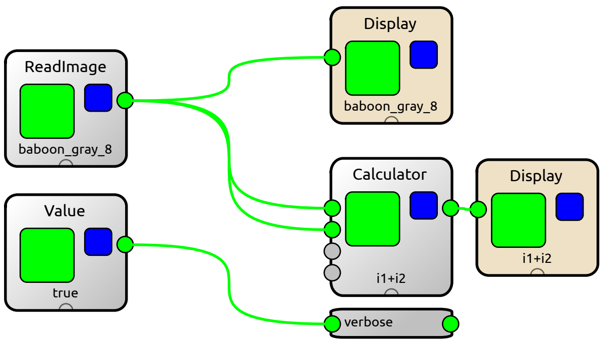


Figure 5.25: ToolIP: Example of the same graph as in Fig. 5.24 but without clamp nodes.

Chapter 6

MAOIcmd

6.1 Usage

MAOIcmd is the command line tool with which one can run a TLP graph once. With **MAOIcmd**, you can run a saved **ToolIP** graph from command line of your computer (Linux or Windows), i.e. without **ToolIP**-GUI and in the command line it is possible to run a graph. The usage of **MAOIcmd** is as follows:

```
MAOIcmd [options] [image_in | value_in]* [parameter]* graph.tlp [image_out |
"_"]*
```

Note that some options are position dependent, that are the ToolIP graph inputs, and the outputs. The position of the TLP graph in the command line option list marks the end of the inputs and the beginning of the outputs. When using a single underscore '_' instead of an output filename, then this result is dropped and not saved. You may use the single underscore '_' to drop inputs from the input list as well, but note that the part of the graph depending on this input will not run, normally the given graph will not run at all.

For value inputs, you need to specify the value type in front of the value. Here you can use:

- `-ps [string]`: pass a string as input
- `-pl [long]`: pass a long integer number as input, signed 32bits or more
- `-pf [float]`: pass floating point value as input, IEEE double precision
- `-pb [bool]`: pass a boolean value as input, proper values are **true** or **false**
- `-pvl [vector of longs]`: pass a vector of long integer numbers as input, comma separated

Parameters are given via `parametername=value`, no spaces allowed, parameter type is determined from the value, that is, values **true** or **false** are leading to a boolean value, numbers are leading to long or float values, etc.

Additional options:

- `-t | -tlp [MAXTHREADS]` : force the method of **ToolIP** graph execution (default) running with MAXTHREADS threads
- `-v | -verbose` : verbose mode prints on standard output execute duration of graph in milliseconds
- `-V | -verbose-level [LEVEL]` : set verbose level for tlp runner, possible values are: silent, critical (default), debug, show_all
- `-top [N]`: print the N slowest running plugins, default N=3.
- `-verbose-output-slot-list` : print map and vector contents in the output slot list
- `-h | -help` : prints detailed help text

Supported image file formats:

- Netpbm *.ppm *.pgm *.pbm *.pfm *.pam: reading, writing
- JPEG, JPEG2000 *.jpg *.jpeg *.jp2 *.j2k: reading, writing
- Bitmap *.bmp: reading, writing
- TIFF, TIFF3d, BigTIFF *.tif *.tiff: reading, writing
- Portable Network Graphics *.png: reading, writing
- Graphics Interchange Format, animated GIF *.gif: reading
- Targa Image File *.tga: partial reading support
- Icons *.ico: reading
- Text, space separated linewise pixeldata *.txt, *.asc, *.ascii, *.csv: reading, writing
- Fraunhofer IASS *.iass *.iass.gz: reading, writing
- CINEfile image sequences: partial reading support

Additional file format support when MAVIkit is available:

- Fraunhofer REK raw volumes: *.rek, *.rek.gz: reading, writing
- GeoDict gdt2 volume data *.gdt
- Raw volume data *.raw: writing support
- AVS/Express Field Data *.fld, *.avs: reading support
- Volume Graphics projects *.vgi, *.vgl: partial reading support
- IMOD MRC image files *.mrc: partial reading support

6.2 Examples

Example 1:

Calling **MAOIcmd** for a graph:

```
MAOIcmd "C:\Program Files\ToolIP\examples\images\baboon.jpg" "C:\Program Files\ToolIP\ex
C:\tmp\baboon_float.tif
```

Note that because the path of the input image included a space, one had to write it in quotation marks.

Example 2:

Running example 1 with the parameter of graph:

```
MAOIcmd C:\Progra~1\ToolIP\examples\images\baboon.jpg method="LIGHTNESS" C:\Progra~1\Too
C:\tmp\baboon_float.tif
```

Example 3:

Calling a graph with value inputs and ignoring the output and with two threads: `MAOIcmd -tlp 2 test.pgm -pf 137.01 -ps qqrq test.tlp _`

Example 4:

In this example, we call a graph for a batch of images. For this purpose we use very simple python code:

```
import os, glob
graph = 'C:\\Progra~1\\ToolIP\\examples\\graphs\\color2gray.tlp'
result_dir = os.getenv('TMP')
img_list = glob.glob('C:\\Progra~1\\ToolIP\\examples\\images\\*.jpg')
for input_img in img_list:
    output_img = os.path.basename(input_img)
    output_img = output_img.replace('.jpg', '_output1.png')
    output_img = os.path.join( result_dir , output_img)
    cmd = ' '.join(('MAOIcmd', input_img , graph, output_img))
    if True: print('This command will be call now:\n', cmd)
    print(os.popen(cmd).read())
```

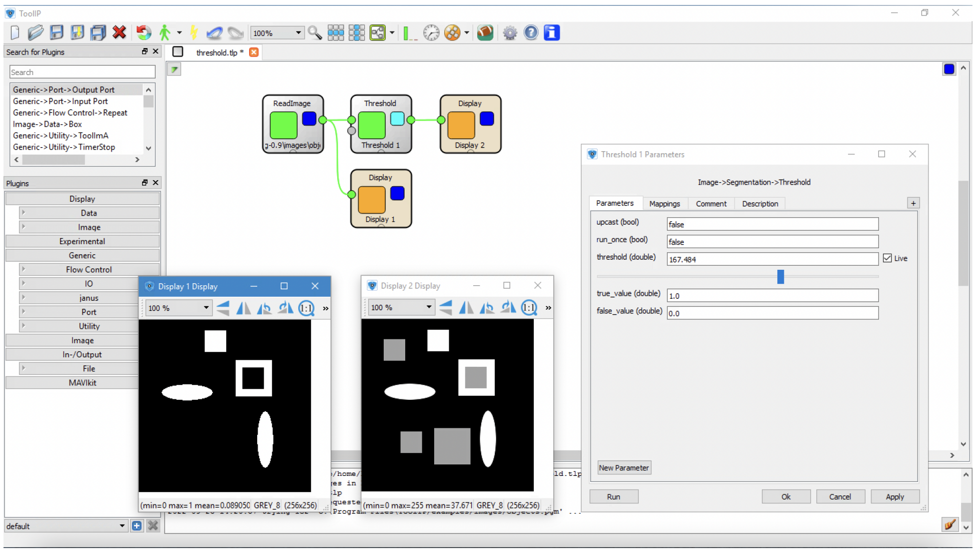
Chapter 7

Further Examples

In this chapter, we take a look at some further examples to point out useful tips and tricks for a better working experience with **ToolIP**.

7.1 Interactive choice of parameters

The first graph we take a look at can be found in the `examples` directory under the name `threshold.tlp` and is displayed in figure 7.1. Concerning the functionality it is very simple: It only performs a threshold on a gray



upcast (bool) false
run_once (bool) false
threshold (double) 167.484 [✓ Live]
true_value (double) 1.0
false_value (double) 0.0

(min=0 max=1 mean=0.089050 GREY_8 (256x256)
(min=0 max=255 mean=37.671 GREY_8 (256x256)

Figure 7.1: Threshold in live mode.

value image. The interesting point with this example is that the **Threshold** node allows to choose the threshold parameter in a live mode. That means there is a slider in the parameter dialog of the threshold plugin that can be dragged with the mouse (see figure 7.2). Since the following display plugin has a bolt symbol (see Chapter 5) this plugin is executed every time the slider is moved. This makes an interactive choice of the parameter possible. If the path of the graph after the plugin with the slider is longer, it might be that the running time is longer and the update of the display slightly slower. Nevertheless, this can still be helpful for efficiently choosing parameters.

7.2 Characterizing objects and textures

It is very common in practical applications that a segmentation yields certain objects that should be characterized geometrically. Now we turn our attention to an example showing how label images can help to do this (see figure

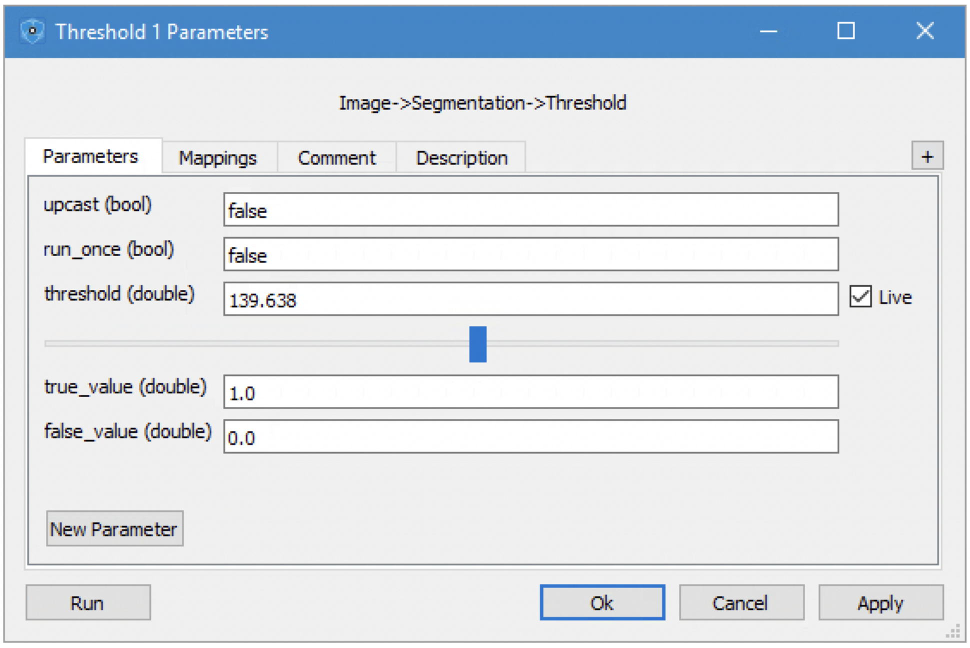

Threshold 1 Parameters

Image->Segmentation->Threshold

Parameters | Mappings | Comment | Description

upcast (bool): false
run_once (bool): false
threshold (double): 139.638 ☑ Live
true_value (double): 1.0
false_value (double): 0.0

Figure 7.2: Parameter dialog of the **Threshold** plugin in live mode.

7.3 and the corresponding graph file `labels.tlp`).

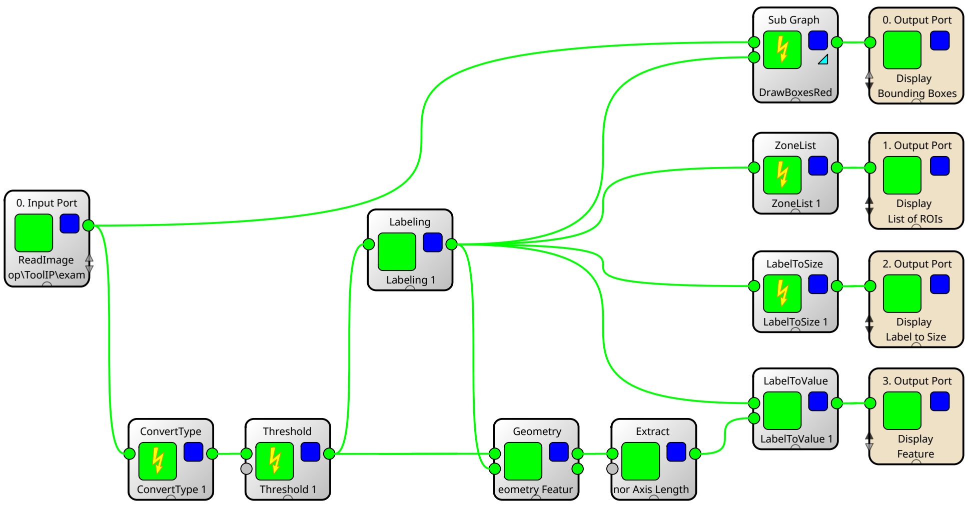


Figure 7.3: Characterizing objects and textures: Example graph.

For simplicity, the segmentation part of the graph is represented by the well-known threshold example from the previous section 7.1. This segmentation yields a binary image where the foreground is marked with 1 and the background with 0. After segmenting the objects, they are now labeled in this example. That means we count the connected components in the foreground, and each pixel inside a connected component is mapped to the number of its component. We now show four ways to extract information out of this labeling:

1. The topmost path uses a subgraph to draw the bounding boxes of all detected objects in red into the input image.

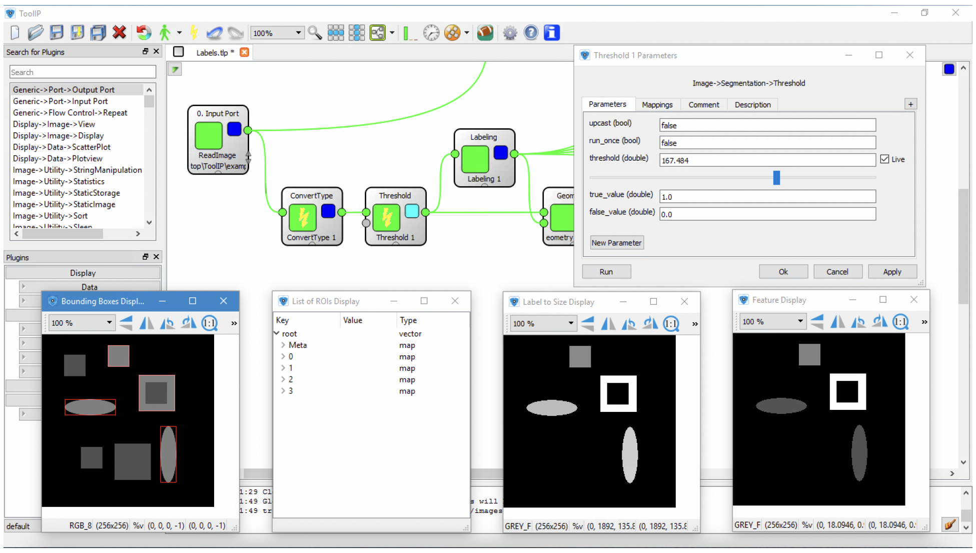
2. The second path creates a **ZoneList** of the objects and displays it as a list.

3. The third path extracts the size of the objects with the **LabelToSize** plugin.

4. The fourth path calculates geometric features of the objects with the **Geometry** plugin and extracts their

minor axis length with the **Extract** plugin. Then it writes the minor axis length in the objects in the image with the **LabelToValue** plugin.

Figure 7.4 shows the results of this graph.



upcast (bool) false
run_once (bool) false
threshold (double) 167.484
true_value (double) 1.0
false_value (double) 0.0

Figure 7.4: Results of running the example graph (see Figure 7.3).

With the fourth path we can see how useful it can be to work with subgraphs: The task for drawing bounding boxes in red might appear more often, and so it is possible to save this subgraph in a separate `.tlp` file (see figure 7.5). We see that **ToolIP** uses in- and output ports to model the inputs and outputs of the subgraph. It is now possible to

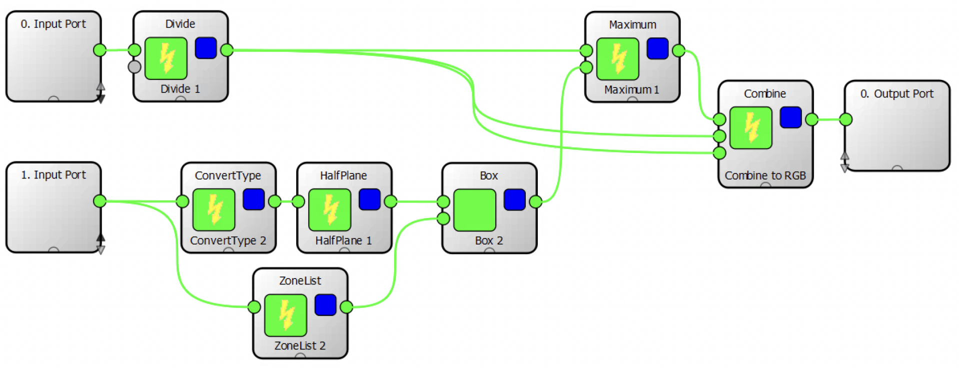


Figure 7.5: Subgraph for drawing bounding boxes in red.

load such a subgraph into the plugin menu by right-click onto the menu and choosing **Load Graphs**. Choosing the `.tlp` file creates a menu entry as shown in figure 7.6. All subgraphs are sorted in under the menu point **Generic** → **Graphs**. This makes it possible to extend the plugin menu by designing graphs for frequently used complex algorithms. This way solutions can be added to **ToolIP**'s algorithms even without writing C++ code.

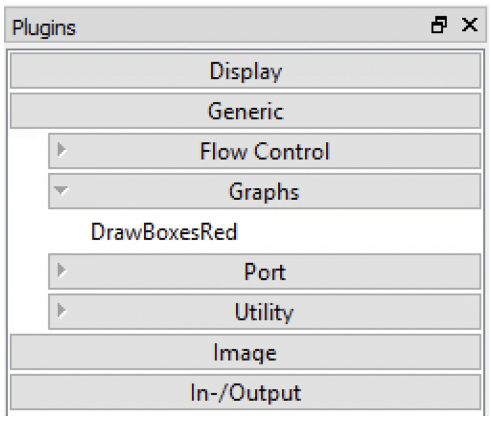


Figure 7.6: Menu entry for the subgraph DrawBoxesRed.

Chapter 8

Generation of Runtime Licenses

In this chapter, we describe the procedure to generate a runtime license file. After doing so, you can run your own **ToolIP** graphs without the dongle and on another PC or notebook. Therefore you could provide others with an image processing application or use your graph solutions by integrating them into your own application framework.

Please note that this is only possible if you own a system builder license for **ToolIP**.
Please note that in all cases this procedure modifies the license file that you are using. So be sure that you did store the original license file safely at another location.

Use the PC or notebook on which you want to be able to use the **ToolIP** graphs. You need to install **ToolIP** first, to do so follow the instructions in section 2 and use your own installer. Use your own license file as well (remember: be sure it is stored safely in another spot!).

After installation put your **ToolIP** Dongle into the USB slot. Wait until it is recognized by the system, then open a command shell as an administrator. You can do this by pressing the windows key and typing "cmd". Then you have to right click on cmd at the top of the opening menu as shown in figure 8.1. Choose the option "Run as administrator".

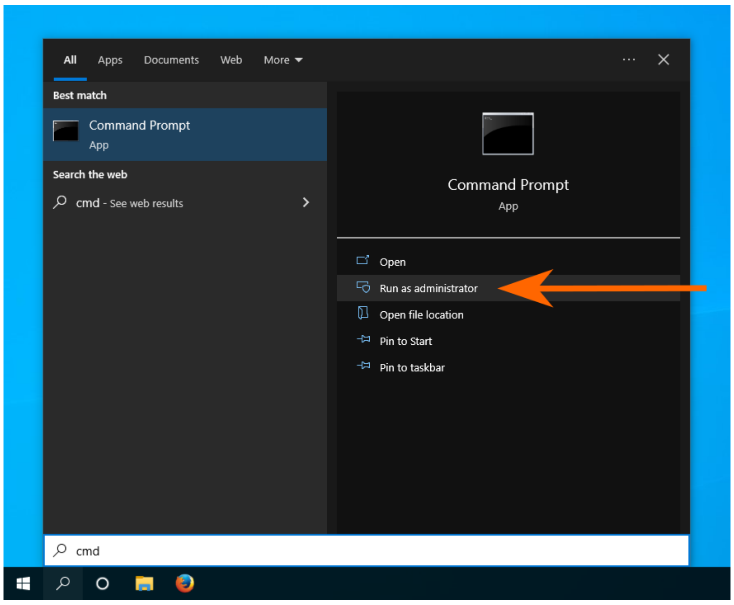
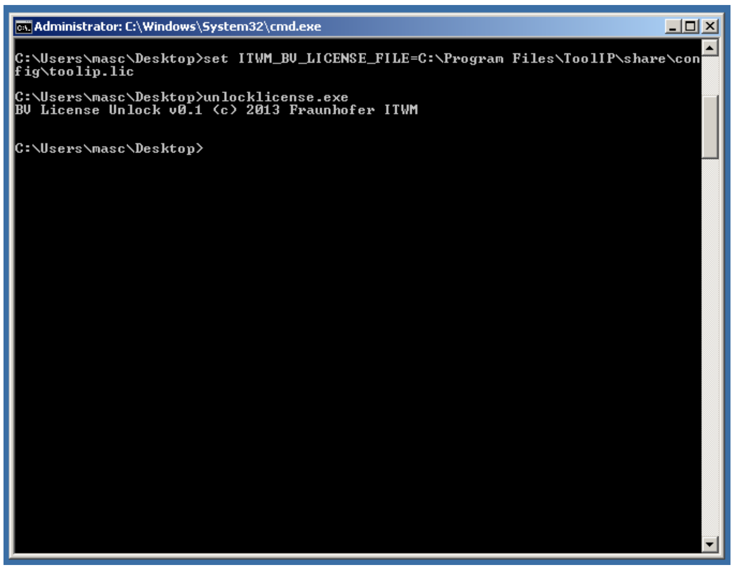


Figure 8.1: Run a command shell as administrator.

In the opened command shell you first have to set the path to your license file. Type

```
set ITWM_BV_LICENSE_FILE=C:\Program Files\ToolIP\share\config\toolip.lic
```

as you can see in figure 8.2 in the first prompt (license file is by default stored at that location). Please beware that there must not be any quotation marks in the file path! Then hit enter.


Administrator: C:\Windows\System32\cmd.exe

```
C:\Users\masc\Desktop>set ITWM_BV_LICENSE_FILE=C:\Program Files\ToolIP\share\con
fig\toolip.lic

C:\Users\masc\Desktop>unlocklicense.exe
BV License Unlock v0.1 (c) 2013 Fraunhofer ITWM


C:\Users\masc\Desktop>
```

Figure 8.2: Set the evironment variable and unlock license file from dongle.

Second you have to type

```
unlocklicense.exe
```

and again hit enter. If there is no error message but the output as in figure 8.2, you are done!

Please note: The license file now is unlocked from your dongle and bound to the current PC or notebook in use. That is, you can run **ToolIP** graphs without the dongle, with that license file and the installed version of **ToolIP** on the current PC or notebook only.

Chapter 9

Algorithm Documentation

9.1 Introduction

Image processing is becoming a crucial technology in a variety of scientific and applied fields. Computer vision, automated quality assurance, biological and medical research, materials science and many others benefit from advanced image processing algorithms.

ToolIP's integrated library provides a large variety of image processing algorithms in a convenient and flexible framework. It has been designed for ease of use and speed. Available algorithms range from simple pixel operations and image filters to advanced data analysis and machine learning methods. State of the art mathematical morphology and linear algebra are also included.

The algorithms are implemented to handle 2D and 3D image data, both gray scale and color types are supported.

This manual is intended as a comprehensive description of the image processing functions provided. Related algorithms are grouped into separate modules (realized technically as different libraries). This structure is reflected in the sections of this manual.

9.2 Toolbox Introductions

9.2.1 Arithmetic

This group of algorithms provides basic arithmetic functions that can be applied either on single images, or to combine the pixel values from two images:

- Operations on single images: These can be used e.g. to add or subtract constants to or from each pixel.
- Operations between two images, e.g., add two images pixel wise to form the result.

A common application example for such pixel wise operations is the correction of shading artifacts: One can apply a large Gaussian filter kernel and subtract the result from the original image. This may help in situations with varying illumination conditions in images. The basic arithmetic operations are supplemented by more advanced functions such as the **RPNC** node (reverse polish notation calculator). This allows to easily apply more complex mathematical expressions to pixel values in one step. In order to invert an image, one may use the **Negative** node, which multiplies every pixel by -1. Furthermore, basic binary valued arithmetic operations are available with the nodes **And** and **Or**.

9.2.2 Classification

Tools and algorithms related to classification can be found in the classification section. By using these nodes, tasks such as the following ones are tackled:

- semantic grouping of segments,
- assigning features vectors to predefined groups,
- detection of statistical outliers.

Next to typical classifiers like **KMeans**, functions such as **PCA** or **ICA** for preprocessing of feature spaces can be found.

Note: The toolbox Classification does not have its own documentation section. You find the node **Convex-Hull** in Chapter Utility Plugins, and the nodes **GaussianMixture**, **ICA**, **KMeans**, **PCA**, and **RANSAC** in Chapter Matrix Plugins.

9.2.3 Color

Most algorithms available in ToolIP deal with gray scale images. Yet, for applications involving color images, the nodes in this section provide basic tools to transform color images from and to gray scale data.

- **Separate**: This node allows to extract single color channels (R, G, B) from a color image and outputs them as gray scale image either in 8-bit or float format.
- **Combine**: This node allows to write three gray scale images into the RGB-color channels of a color image.
- **ColorTransform**: Converts an RGB-color image into various other color formats such as HSV or YUV.

Other than the transformations themselves, especially the node **ColorTransform** can be useful to identify a color representation which sometimes allows e.g. for more accurate segmentation.

9.2.4 Data

The data module provides a set of tools for parametrized image creation. This can range from simple geometrical objects like balls, boxes and lines to more sophisticated ones like splines. Random numbers according to different distributions can be generated too.

By combining different data plugins, parametrized sequences of test images can be easily created. Another important use is the creation of adaptive masks to facilitate segmentation or the visual presentation of the final result of for example a detection algorithm. Still another use is the generation of realization of stochastic geometry models like for example the boolean model.

9.2.5 Display

Display groups together all nodes available for data and image visualization. Thus, the nodes found here are all terminals. The following display nodes are currently available:

- **View** provides fast and simple viewing of 2D and 3D images.
- **Display** is a highly configurable node that allows to display images and data contained in value formats such as ValueMap or ValueDouble.
- **PlotView** is a tool for displaying 1D functions and histograms.

9.2.6 Features

The features module provides tools to characterize image regions. Regions are defined by assigning the same label value to a set of pixels, i.e. pixels with the same label belong to one region. The most common case is that regions correspond to connected components in a binary image, then the label assignment is performed by the node **Labeling** producing a label image.

Image regions can be characterized by their geometry (shape of the region) or by their content (values of the pixels they contain). Features like form factor, area, boundary length and orientation deal with the geometry aspect. Features like average, variance, entropy or the Haralick features deal with gray values, their distribution and spacial arrangement.

Once a characterization of the regions has been obtained, further algorithms like clustering or classification can be applied to detect common features or classify the regions according to specific criteria.

9.2.7 Filter

The idea of filtering is to replace the value of a pixel by combining the values in its neighborhood. How the neighborhood is defined depends on the specific filter. The filter function can be a linear combination of input values (linear filter) or a more general nonlinear function.

Filtering is typically used in the initial steps of the processing chain. It is a powerful technique for image corrections, image restoration and denoising. Moreover, special filters can be used to detect and characterize relevant structures, as for instance edges, corners or local extrema. Applying some filtering can also be helpful to facilitate segmentation or analysis at later stages.

The library provides several standard linear (e.g. Average, Gauss) and nonlinear (e.g. Median) filters as well as filters based on partial differential equations (PDEs).

9.2.8 LabelImage

In various applications, an image contains more than one region of interest. It is often useful to consider these regions separately. This is possible with the LabelImage plugin group. One of the most important nodes is **Labeling**, where labels are created from a binary image by incrementally numbering all connected foreground components. Further plugins from this group can be used to assign color, size, or specific values to these regions.

Another frequently used node from the current section is **FindMax**, which can be used to extract the parameters of lines and circles when combined with **Hough** and **HoughCircle**.

Note: The toolbox LabelImage does not have its own documentation section. You find the nodes **BoundingBox**, **FindMax**, **FindBalancePoint**, **Labeling**, **ZoneList** in Chapter Utility Plugins, the node **Hist** in Chapter Features Plugins and the nodes **LabelToColor**, **LabelToFeature**, **LabelToSize**, **LabelToValue** in Chapter Manipulation Plugins.

9.2.9 Manipulation

The manipulation group of nodes contains various tools to manipulate either the coordinate system or the observation window of an image. The range of functionality of these nodes spans from simple concepts such as translation or rotation up to non-linear coordinate transformations such as **CartesianToPolar**.

The manipulation nodes find their applications as subroutines in almost all image processing graphs. They are very versatile, which renders them suitable to create highly adaptive and robust graphs.

9.2.10 Matching

Matching or registration is the process of transforming the given data into one common coordinate system. This is usually necessary to allow comparison of image data acquired in different coordinate systems at different times, or using different imaging techniques. It can also be used to detect known objects in an image or for motion estimation.

9.2.11 Matrix

Images can be interpreted as matrices, where each pixel value corresponds to a matrix entry value in a natural way. Image characteristics can be extracted in form of features and stored as an image. This allows to apply linear algebra methods to process and analyse features extracted from an image without the need of an additional external library.

The matrix module provides many standard linear algebra operations (e.g. Multiply, SVD, Eigenvalues) that can be combined to realize more complex algorithms (e.g. Spectral Clustering, Kernel PCA).

9.2.12 Morphology

Mathematical morphology encompasses a range of non-linear filter operations with respect to a filter mask, which is called “structuring element” within this context. In morphology, the maximum and minimum filters correspond to the so-called Dilation and Erosion operators, respectively. The resulting functions can be used to correct image artifacts, to reconstruct image structures lost during image acquisition, to detect edges and other predefined structures, or to prepare images for segmentation.

Starting from those basic building blocks, various image processing operations are available, e.g.:

- **Opening** and **Closing** to separate and to join image regions or structures, respectively.
- **FillHoles** and **CutHills** to remove isolated gray value “valleys” and “peaks”.
- **BlackTopHat** and **WhiteTopHat** to emphasize dark and bright regions at least as large as the given structuring element.

The shape of the structuring elements is restricted to rectangles (2D) and cuboids (3D). In all three coordinate directions, its size must be specified by the half lengths.

Note: By definition, the resulting structuring element will always be of size $2n+1$ for a given parameter value $n > 0$.

9.2.13 Segmentation

Segmentation is the process of partitioning the image into regions. How the regions are defined depends on the image data or on the intended application. Result of the segmentation of a grey value image can be a binary image, i. e. where the regions consist of background and foreground pixels, foreground usually corresponding to the object of interest. A more complex example would be to identify every single object in a collection resulting in a label image.

ToolIP provides threshold based segmentation procedures (e.g. Otsu) as well as more sophisticated algorithms (e.g. ChanVese).

9.2.14 Transformation

This section describes mathematical, reversible transformations between different bases, special transformations suitable for feature extraction as well as a few utilities that enable the handling of the resulting data. Application examples are:

- **Hough** and **HoughCircle** can be used to locate lines and circles in images, respectively.
- Wavelet transformations (**OrthoFWT** and **OrthoIWT**) are frequently applied in multi-scale image processing.
- Fourier transformations (**FFT**) are useful in spectral image analysis, e.g. for texture analysis.

In contrast to most other modules in **ToolIP**, these transformations operate not in the usual cartesian space.

Note: The toolbox Transformation does not have its own documentation section. You can find the node **Distance** in Chapter Filter Plugins, the node **FFT** in Chapter FFT Plugins, the nodes **Hough**, **HoughCircle**, **Radon** in Chapter Utility Plugins, the nodes **ModulusMaxima**, **MultiscaleWavelet**, **OrthoFWT**, **OrthoIWT** in Chapter Wavelets Plugins.

9.2.15 Utility

The methods in this group can be described as auxiliary functions. One of the most frequently used is **ConvertType**. This plugin converts the input image into another format that can be used by the next plugin - this is necessary as most of them require a specific image type input (IMAGE_GREY_F is the most common one). Some of them are characteristics that are calculated over the whole image (in contrast to the feature plugin group), such as **Statistics**, **Histogram** (1 and 2D), **MutualInformation**, and more. Others introduce or change the content of the image, such as **Normalize**, **Projection**, **PaintMarks**, and so on.

9.3 Arithmetic Plugins

9.3.1 Plugin Abs

The absolute value, pixel wise.

Computes the absolute value of the input image pixel wise.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type and size as the input image.

Keywords:
absolute, value, arithmetic

9.3.2 Plugin AbsDiff

Node to compute the absolute value of the pixel wise difference of two images.

Computes the absolute value of the pixel wise difference of two images.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Second input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as the input images. For mixed input image types or with *upcast* = *true*, the output type is IMAGE_GREY_F. The size of the output image is the minimum of the respective input image sizes.

Parameters

upcast(bool)	Indicates whether the output image is cast to float. By default it is set to false.

Keywords:

arithmetic, difference, absolute value

9.3.3 Plugin Add

Add images pixel wise or add a constant to an image.

Add the two input images (if second input is available) or add the constant *add* to the first image (if second input is not available).

- **Inpin 1:** Input image of type *IMAGE_GREY_F* or *IMAGE_GREY_8*.
- **Inpin 2:** Optional second input image of type *IMAGE_GREY_F* or *IMAGE_GREY_8*.

- **Outpin 1:** Output image of the same type as input image(s). For mixed input image types the output type is *IMAGE_GREY_F*. In case of two input images the size of the output image is the minimum of the respective input image sizes.

Parameters

add(double/long)	This constant value will be added to the first input image in case only one input image is provided. By default it is set to *0.0*.
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.

Keywords:

addition, sum, plus, arithmetic

9.3.4 Plugin And

Node to apply the logical AND operation on two images of the same size.

Applies the logical AND operation on two images of the same size. If no second image is given, the first image and an arbitrary value of the parameter *bitmask* will be concatenated by AND.

- **Inpin 1:** Input images of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Input images of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as the input images.

Parameters

bitmask(string)	Binary or hexadecimal string applied to each pixel in case no second input is given. By default, it is set to *0xFFFF* (no change).
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.
run_flag(bool)	INTERNAL Indicates whether the node has already run before. By default it is set to *false*.

Keywords:

arithmetic, logical, boolean, bitwise, bitmask

9.3.5 Plugin AssertEq

Plugin compares two images pixel-wise and raises an error if they do not match.

When two input images are given, the plugin compares them pixel-wise. If they don't match, the plugin raises an error with the text given by parameter 'error_message' and some additional information.

- **Inpin 1:** Input image of type IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, IMAGE_GR↩ EY_F, or BINARY_FG.
- **Inpin 2:** Second input image. Type and size must match the first input image if parameter 'ignore_↩ imagetype' is FALSE, else only size must match.

- **Outpin 1:** If no error occured, the output will be the string value of 'error_message'. This is helpful for triggering plugins if and only if the two input images are equal (with respect of 'epilson').

Parameters

error_↩ message(string)	Error message prepend string. By default it is set to "Error".
epsilon(double)	Sets numerical precision for the comparison. More precisely, if *epsilon* $>$ 0, the compared values are considered as equal if their absolute difference is smaller than *epsilon*. By default it is set to 0.
ignore_↩ type(bool)	If TRUE, image types are ignored and only the pixel values are checked for equality. If FALSE, input images must be of same type.
negate_↩ assert(bool)	If FALSE (default), raises error if images are not equal. If TRUE, raises error if images are equal.
verbose(bool)	Determines whether some information is printed out on the screen.

Keywords:

throw, error, assert, condition, comparison, raise error, equal, same

9.3.6 Plugin Ceil

Round up each pixel value.

Plugin computes the smallest integral value which is not less than the actual value for every pixel of the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of same type and size as the first input image.

Parameters

upcast(bool)	Whether the output image is to be cast to float. By default it is set to *false*.

Note

Rounds UP every pixel value to the nearest integer.

See also

plugin *Floor*

Keywords:

ceil, round down, integer number

9.3.7 Plugin Cos

Plugin to compute the cosine value cos(x) for every pixel value x from the input image.

Computes the cosine value cos(x) for every pixel value x from the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of same type and size as the first input image.

Parameters

upcast(bool)	Whether the output image is to be cast to float. By default it is set to *true*.

Keywords:

cos, cosine, trigonometric function, trigonometry

9.3.8 Plugin Divide

Node to divide two images pixel wise or to divide the input image by a scalar or a scalar by the input image values.

If two input images are given, they are divided pixel wise. If only one input image is given, the input image is divided by the given scalar or vice versa.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as input image(s). For mixed input image types output type is always IMAGE_GREY_F. In case of two input images the size of the output image is the minimum of the respective input image sizes.

Parameters

order(bool)	If only one input image is given, the result will be the pixel wise quotient of the pixel values and the scalar parameter *divide*. If order equals 0, result = image / *scalar*, otherwise result = *scalar* / image. By default it is set to 0.
di-vide(double/long)	This input parameter is only used if just one input image is given. It must be a constant and the input image will be divided by it if parameter *order* equals 0. Otherwise the scalar *divide* is divided pixel wise by the image values. It will implicitly be cast to double and is set to 1.0 by default.
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.
divbyzeroiser-ror(bool)	Division by zero is treated as error. If set to false, a division by zero results for float images in +inf or -inf (or nan if input is nan); for integer images it results in 0. By default it is set to *true*, i.e. division by zero is treated as error.

See also

plugin ReplaceNonFinite which is useful when division results in non-finite pixel values

Keywords:

divide, pixelwise division, divisor, dividend

9.3.9 Plugin Equal

Plugin to compare two images pixel wise or an image and a scalar.

Compares two images pixel wise or an image and a scalar. Results in an image holding two pixel values. One is indicating equality, the other one inequality. If two input images are given, they will be compared pixel wise. If the pixel values match, the result at this position is set to the value given by parameter *true_value*. Otherwise the result at this position is set to the value given by parameter *false_value*. If only one input image is given, each pixel will be compared to the parameter *equal*.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image of type IMAGE_GREY_F or IMAGE_GREY_8 and must be of the same size as first input image.

- **Outpin 1:** Output image of the same type and size as the first input image(s). If parameter 'upcast' is TRUE, then the result is casted to the 'higher' type, that is, for mixed input types, the result type is IMAGE_GREY_F.

Parameters

equal(double/long)	This parameter is only used if just one input image is given. It must be a constant and the input pixel values will be compared to it. It will implicitly be cast to double and is set to 0.0 by default.
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.
epsilon(double)	Sets numerical precision for the comparison. More precisely, if *epsilon* > 0, the compared values are considered as equal if their absolute difference is smaller than *epsilon*. By default it is set to 0.
true_↩ value(double/long)	Pixel value in output image indicating equality. It is cast to double. By default it is set to 1.0.
false_↩ value(double/long)	Pixel value in output image indicating inequality. It is cast to double. By default it is set to 0.0.

Keywords:

equal, compare, comparison, ==, isequal

9.3.10 Plugin Exp

Plugin to compute the exponential value for every pixel value from the input image.

Computes exp(x), that is e to the power of x, where x will be every pixel value from the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of same type and size as the first input image.

Parameters

upcast(bool)	Whether the output image is to be cast to float. By default it is set to *true*.

Keywords:

exp, exponential function, natural function

9.3.11 Plugin Floor

Round down each pixel value.

Plugin computes the largest integral value which is not greater than the actual value for every pixel of the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of same type and size as the first input image.

Parameters

upcast(bool)	Whether the output image is to be cast to float. By default it is set to *false*.

Note

Rounds DOWN every pixel value to the nearest integer.

See also

plugin *Ceil*

Keywords:

floor, round up, integer number

9.3.12 Plugin Log

Plugin to compute the natural logarithm log(x) for every pixel value x from the input image.

Computes the natural logarithm log(x) for every pixel value x from the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of same type and size as the first input image.

Note

Depending on the input pixel values and used function, resulting pixel values can be infinity or NAN.

Parameters

upcast(bool)	Whether the output image is to be cast to float. By default it is set to *false*.

Keywords:

log, logarithm, natural logarithm

9.3.13 Plugin Math

Plugin to compute various mathematical functions on the input image pixel wise.

Compute various mathematical functions on the input image pixel wise. Provides logarithmic, exponential, power, rounding, and various trigonometric functions.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional input image of type IMAGE_GREY_F or IMAGE_GREY_8. Needed as second argument for *pow* and *atan2*.

- **Outpin 1:** Output image of same type and size as the first input image.

Note

Depending on the input pixel values and used function, resulting pixel values can be infinity or NAN.
Executes pixel wise exactly the function with the same name from math and math2 C++ headers.
ASINH, ACOSH, and ATANH are not available on Windows.

Parameters

function(string)	Shortname of the mathematical function to be applied. Available functions are *LOG*, *EXP*, *POW*, *SIN*, *ASIN*, *COS*, *ACOS*, *TAN*, *ATAN*, *ATAN2*, *SINH*, *COSH*, *TANH*, *ASINH*, *ACOSH* and *ATANH*, *CEIL* and *FLOOR*. By default it is set to *LOG*.
upcast(bool)	Whether the output image is to be cast to float. By default it is set to *false*.

Keywords:

math, math functions, trigonometric, power, logarithmic, round

9.3.14 Plugin MathConstant

Plugin provides various mathematical constants.

Provides various mathematical constants, as value or image filled with. If an input image is available, the output is an image of type IMAGE_GREY_F filled with the chosen constant. If no input image is available, the output is a value of type DOUBLE with value of the choosen constant.

- **Inpin 1:** Optional input image, controls type of output.

- **Outpin 1:** The constant or constant image.

Parameters

constant(string)	Indicates the chosen mathematical constant: • *PI*, • *PI_HALF*, • *PI_THIRD*, • *PI_FOURTH*, • *PI_THREE_FOURTH*, • *TWO_PI*, • *E*, • *SQRT_TWO*, *SQRT_THREE*, • *INF*, *INF_NEG*, *NAN* • *GOLDEN_RATIO* By default it is set to *PI*.

Keywords:

pi, e, infinity, nan, not-a-number, mathematical constants

9.3.15 Plugin Maximum

Node to compute the maximum of two images pixel wise or the maximum of the pixel values and a scalar.

Computes the maximum of two images pixel wise or the maximum of the pixel values and a scalar.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as input image(s). For mixed input image types output type is always IMAGE_GREY_F. In case of two input images the size of the output image is the minimum of the respective input image sizes.

Parameters

upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.
maximum(double)	If only one input image is given, this is the scalar value to compare with. By default it is set to 255.0.

Keywords:

maximum, max, compare

9.3.16 Plugin Minimum

Node to compute the minimum of two images pixel wise or the minimum of the pixel values and a scalar.

Computes the minimum of two images pixel wise or the minimum of the pixel values and a scalar.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as input image(s). For mixed input image types output type is always IMAGE_GREY_F. In case of two input images the size of the output image is the minimum of the respective input image sizes.

Parameters

upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.
minimum(double)	If only one input image is given, this is the scalar value to compare with. By default it is set to 255.0.

Keywords:

minimum, min, compare

9.3.17 Plugin Modulus

Node to compute the modulus operation pixel wise on two images or on the pixel values of an input image and a scalar.

If two input images are given, the pixel wise modulus operation will be performed (divisor image1 mod dividend image2). If no second input image is given, the modulus is calculated on the input image with the value of the parameter *modulo* as second argument.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8. If *order* == 0, then this is the dividend, else the divisor.

- **Inpin 2:** Optional second input image of type IMAGE_GREY_F or IMAGE_GREY_8 and same size as first input image.

- **Outpin 1:** Output image of the same type as input image(s). For mixed input image types output type is always IMAGE_GREY_F.

Parameters

order(bool)	If only one input image is given, the result will be the pixel wise modulus of the pixel values and the scalar parameter *modulo*. If order equals 0, result = image `%` *scalar*, otherwise result = *scalar* `%` image. By default it is set to 0.
modulo	(Divisor) value to perform the pixel wise modulus operation with, if only one image is given. If *order* != 0 it is the dividend value. It will be cast to long. By default it is set to 1.0.
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.

Keywords:

modulus, residual, modulo, mod

9.3.18 Plugin Multiply

Multiply two images pixel wise, or an image with a scalar value.

Multiplies two images pixel wise or an image and a scalar value. If two input images are given, the product of these two will be computed pixel wise. If no second image is given, the image is multiplied pixel wise with the value of the parameter *multiply*.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as input image(s). For mixed input image types output type is always IMAGE_GREY_F. In case of two input images the size of the output image is the minimum of the respective input image sizes.

Parameters

upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.
multi-ply(double/long)	Value to multiply the input image with, if only one image is given. It will be cast to double. By default it is set to 1.0.

Keywords:

multiply, pixelwise multiply, mult, factor

9.3.19 Plugin Negative

Node to compute the negative of the input image pixel wise.

Computes the negative of the input image pixel wise. The input image gets subtracted pixel wise from the parameter value *shift*.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of same type and size as input image.

Note

If the input image is of type IMAGE_GREY_8, the subtraction is done with respect to 256 possible gray values [0 to 255].
If the input image is of type IMAGE_GREY_8, *shift* is rounded down to the next integer.

Parameters

shift(double/long)	Value to subtract the pixel values from. It is cast to double. By default it is set to 0.0, thus only the sign is changed.
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.

Keywords:

negative, invert

9.3.20 Plugin Or

Node to apply the logical OR operation on two images of the same size.

Applies the logical OR operation on two images of the same size and of type IMAGE_GREY_F or IMAGE_GR↩ EY_8. If no second image is given, the first image and an arbitrary value of the parameter *or* will be concatenated by OR. By default *or* is set to 0, so the input image is not changed.

Note

Input images must have the same size and must be of type IMAGE_GREY_F or IMAGE_GREY_8.

Parameters

or(long)	sets the parameter to compute the bitwise OR with, if no second image is given. It will implicitly be cast to long. By default, *or* is set to 0, so the input image is not changed.
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.

Keywords:

or, or operation, logical operation

9.3.21 Plugin Pow

Plugin to compute the power every pixel value from the input image.

Computes y=pow(x,p), that is x to the power of p, where x will be every pixel value from the input image and y the pixel value of the new image.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8, the power.

- **Outpin 1:** Output image of same type and size as the first input image.

Parameters

upcast(bool)	Whether the output image is to be cast to float. By default it is set to *true*.

Keywords:

pow, power

9.3.22 Plugin RPNC

Calculator on images in Reverse Polish Notation Calculation (RPNC) notation.

Applies a Reverse Polish Notation Calculation (RPNC) pixel wise on the input images. The following operations and special tokens are supported:

- *i1* - first input image
- *i2* - second input image, optional
- *i3* - third input image, optional
- *cp* - copy top entry
- *rm* - remove top entry
- *x* - x pixel coordinate (first image)
- *y* - y pixel coordinate (first image)
- *z* - z pixel coordinate (first image)
- *w* - width (first image)
- *h* - height (first image)
- *d* - depth (first image)
- +, -, $*$, /, %, min, max - add, subtract, multiply, divide, modulo, minimum, maximum
- $>$ - greater than
- $<$ - less than
- == - equal
- != - not equal
- ?: or *IFELSE* - conditional selection, trinary: if(condition) then value1 else value2
- *sqrt* or SQRT - square root
- *abs* or ABS - the absolute value
- *LOG* - natural logarithm
- *EXP* - exponential function e
- *SIN* - sine of x, x given in radians
- *ASIN* - arc sine of x in radians
- *COS* - cosine of x, x given in radians
- *ACOS* - arc cosine of x in radians
- *TAN* - tangent of x, x given in radians
- *ATAN* - arc tangent of x in radians
- *SINH* - hyperbolic sine of x
- *COSH* - hyperbolic cosine of x
- *TANH* - hyperbolic tangent of x
- *ASINH* - inverse hyperbolic sine of x
- *ACOSH* - inverse hyperbolic cosine of x

- *ATANH* - inverse hyperbolic tangent of x
- *NEG* - negation operator, that is -x
- *ATAN2* - arc tangent of y/x using the signs of the two arguments to determine the quadrant of the result
- *POW* - power, binary function x^y
- *CASTB* - cast image to boolean type IMAGE_MONO_BINARY: pixel value 0 -> 0, pixel value not 0 -> 1
- *CAST8* - cast image to 8bit unsigned integer type IMAGE_GREY_8, under/overflow may occure
- *CAST16* - cast image to 16bit unsigned integer type IMAGE_GREY_16, under/overflow may occure
- *CAST32* - cast image to 32bit unsigned integer type IMAGE_GREY_32, under/overflow may occure
- *CASTF* - cast image to floating point type IMAGE_GREY_F
- Numerical constants like 1.0, 127, ...
- Numerical parameters by their names
- Single input/output image graphs (XML and TLP) by their path
  Graph parameters can be specified by giving the key followed by '=' and followed by the value, example: angle=17.0 . If the parameter value contains spaces it must be enclosed in single quotes: image='name of image.jpg'.

- **Inpin 1:** Input image or a value of type bool, long, or float
- **Inpin 2:** Optional second input image
- **Inpin 3:** Optional third input image

- **Outpin 1:** If the last operation is not a casting operation, output image of the same type as input image(s). For mixed input image types output type is always IMAGE_GREY_F. If the last operation is a cast, the output gets this type, see CAST∗ operators above. In case of two input images the size of the output image is the minimum of the respective input image sizes.

Note

Values enclosed in quotes are always treated as strings.
User can add new parameters to the plugin. In the *expression* strings, those can be then used. Therefore, operator names and special characters/character sequences like "=" and "::" are forbidden for parameter names.
Reserved characters

- "::", "=", space, tabs, any type of brackets or operators - do not use those characters/character sequences in mapped RPNC parameters

Parameters

expression(string)	Specifies what calculations are applied on the input data. Reverse polish notation syntax is used, so "2 2 + 3 ∗" signifies (2+2)∗3. By default *expression* is set to *i1*.
divbyzeroiserror(bool)	Division by zero is treated as error. If set to false, a division by zero results for float images in +inf or -inf (or nan if input is nan); for integer images it results in 0. By default it is set to *true*, i.e. division by zero is treated as error.

Keywords:

RPNC, RPN, calculator, arithmetic, reverse polish notation

See also

Calc plugin

9.3.23 Plugin Sin

Plugin to compute the sine value sin(x) for every pixel value x from the input image.

Computes the sine value sin(x) for every pixel value x from the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of same type and size as the first input image.

Parameters

upcast(bool)	Whether the output image is to be cast to float. By default it is set to *true*.

Keywords:

sin, sine, trigonometric function, trigonometry

9.3.24 Plugin SquareRoot

Computes the pixelwise square root on the input image.

This plugin computes the square root of the input image pixel wise.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type and size as input image.

Parameters

upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.

Note

Negative input pixel values cause the resulting value to be NaN (not a number).

Keywords:

square root

9.3.25 Plugin Subtract

Pixelwise subtraction of one image from another or subtraction of a scalar from the input image or the input image from a scalar.

If two input images are given, the second one is subtracted from the first one pixel wise. If only one input image is given, the difference between the pixel values and the given scalar is computed.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as input image(s). For mixed input image types output type is always IMAGE_GREY_F. In case of two input images the size of the output image is the minimum of the respective input image sizes.

Parameters

order(long)	If only one input image is given, the result will be the pixel wise difference between the pixel values and the scalar parameter *subtract*. If *order* equals 0, result = image - *scalar*, otherwise result = *scalar* - image. By default order is set to 0. Note: *order* may be also of type bool or double
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to *false*.
sub-tract(double/long)	This input parameter is only used if just one input image is given. It must be a constant and will be subtracted from the input image if parameter *order* equals 0. Otherwise the image is subtracted pixel wise from the scalar value *subtract*. It will implicitly be cast to double and is set to 0.0 by default.

Keywords:

subtract, subtraction, minus, arithmetic

9.3.26 Plugin Tan

Plugin to compute the tangent tan(x) value for every pixel value x from the input image.

Computes the tangent tan(x) value for every pixel value x from the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Outpin 1:** Output image of same type and size as the first input image.

Parameters

upcast(bool)	Whether the output image is to be cast to float. By default it is set to *true*.

Keywords:

tan, tangent, trigonometric function, trigonometry

9.4 Barcode Plugins

9.4.1 Plugin BarcodeReader

Node for decoding barcode/QR code from an image based on the thirdparty library ZBar.

This node uses the ZBar library to decode barcodes/QR codes in the input image. The recognized characters are written into the output as a string.

- **Inpin 1:** Image of type IMAGE_GREY_8.
- **Outpin 1:** Recognized characters from the barcode as a string.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

Credits:

This plugin uses the ZBar library. For additional information, see `Third Party License`

Keywords:

barcode, qr, zbar, decoding, decode

9.5 Color Plugins

9.5.1 Plugin AlphaChannel

Alpha channel manipulation.

- **Inpin 1:** color image
- **Inpin 2:** alpha channel, optional
- **Outpin 1:** color image, alpha channel manipulated.

9.5.2 Plugin ColorMap

Coloring grayvalue images.

Colorize the input image using the given *colormap*.

- **Inpin 1:** grayvalue image

- **Outpin 1:** colorized image

Parameters

colormap(string)	colormap to be used. Available colormaps are: • RED • GREEN • BLUE • GREY • DISCRETE • HOT • BENE • RENE • CYCLIC • LOAD_FROM_FILE

•

Keywords:
color, colour, colorize, colormap

9.5.3 Plugin ColorTransform

Node to separate RGB-images and to decompose them into various different color spaces.

An input image is transformed into three channels that are written into three output images. The output images contain the information of three different selectable color channels.

- **Inpin 1:** Input image of type IMAGE_RGB_8 or IMAGE_RGB_8I.

- **Outpin 1:** First channel as IMAGE_GREY_F depending on given color mode.

- **Outpin 2:** Second channel as IMAGE_GREY_F depending on given color mode.
- **Outpin 3:** Third channel as IMAGE_GREY_F depending on given color mode.

Parameters

color_↩ type(string)	Determines the color space the input image is decomposed to. Possible inputs are *RGB*, *CMY*, *YUV*, *YCbCr*, *YIQ*, *HSV*, *HSI*, *I1I2I3*, *XYZ* and *Lab*. By default it is set to *RGB*.

Note

The range of the output values varies with the chosen color space. In some cases it might be necessary to normalize the result in order to make it visible in the output image.

Keywords:

color transform, RGB, HSI, CMY, YUV, YCbCr, YIQ, I1I2I3, XYZ, Lab, HSV, channel, color channels

9.5.4 Plugin Combine

Node to compose three images to build an RGB-image.

Three equally sized input images are used as three color channels and get combined to an RGB output image.

- **Inpin 1:** First channel input image of type IMAGE_GREY_8 or IMAGE_GREY_F.
- **Inpin 2:** Second channel input image of type IMAGE_GREY_8 or IMAGE_GREY_F.
- **Inpin 3:** Third channel input image of type IMAGE_GREY_8 or IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_RGB_8 or IMAGE_RGB_8I, determined by parameter *output↩ _type*.

Parameters

output_↩ type(string)	Defines the type of the output image, possible entries are *IMAGE_RGB_8* and *IMAGE_RG↩ B_8I*. By default it is set to *IMAGE_RGB_8*.
verbose(bool)	Prints function feedback to log area. By default it is set to *false*.
input_color_↩ type(string)	Sets the input color type. The types *RGB*, *CMY*, *YUV*, *YCBCR*, *YIQ*, *I1I2I3* and *XYZ* are supported. By default the value is set to *RGB*.

Keywords:

combine colors, RGB, CMY, YUV, YCBCR, YIQ, I1I2I3, XYZ, channel, color channels

9.5.5 Plugin Separate

Node to separate an image into its channels.

The input image has to be an RGB image whose color channels get separated and are written to three output images: one for the R, one for the G, and one for the B color channel.

- **Inpin 1:** Input color image of type IMAGE_RGB_8I or IMAGE_RGB_8.

- **Outpin 1:** Red channel output image of type IMAGE_GREY_8 or IMAGE_GREY_F, determined by parameter *out_type*.

- **Outpin 2:** Green channel output image of type IMAGE_GREY_8 or IMAGE_GREY_F, determined by parameter *out_type*.
- **Outpin 3:** Blue channel output image of type IMAGE_GREY_8 or IMAGE_GREY_F, determined by parameter *out_type*.

Parameters

out_type(string)	Determines the type of the output images. Possible types are *IMAGE_GREY_8* or *IMAGE↩ _GREY_F*. By default it is set to *IMAGE_GREY_8*.

Keywords:

separate, channel, RGB, CMY, YUV, YCBCR, YIQ, I1I2I3, XYZ, color channels, split

9.6 Calculator Plugins

9.6.1 Plugin ValueCalc

Expression Calculator.

A simple calculator plugin.

- **Inpin 1:** Input image or value referred to as `i1`.
- **Inpin 2:** Input image or value referred to as `i2`.
- **Inpin 3:** Input image or value referred to as `i3`.
- **Inpin 4:** Input image or value referred to as `i4`.

- **Outpin 1:** Result of the calculation.

Parameters

expres-sion(string)	The arithmetic expression to compute.
verbose(bool)	Determines whether some run information should be printed onto the screen. By default it is set to false.

Note

Most functions in this calculator do only work with **IMAGE_GREY_8** and **IMAGE_GREY_F** images.

9.6.1.1 Examples

`i1, i2, i3, i4` are the variables representing the input ports. Here are some expressions:

- `i1 * 2`
- `i1 * i1 * i3`
- `i1.x`
- `i1.y`
- `i1.z`
- `i1.width`

- `if(i1, 10, 0)`
- `if(i1 > 42.0, 1, 0)`
- `i2 + 3 * pi^5 - grey8(i1)`
- `5*"Hello World"`
- `assert(1 == 0, i1, "1 is not equal to 0 !")`
- `assert(1 == 1, i1, "You will never see this error message.")`

The plugin supports numbers of type double or long, boolean values, strings and images. A value of type double is a floating point value with double precision, a value of type long is a signed integer of at least 32bit.

The calculator performs automatic upcasts when we cannot be sure the target type is suitable for the result of a computation. This means, that for example the result of `pow(i1, 4)` is an IMAGE_GREY_F, because we cannot assert that all values of `i1^4` are less than 255 and can therefore fit in an IMAGE_GREY_8. If you need a specific data type, you can always use one of the cast functions `string(x), bool(x), float(x), int(x), grey8(x), grey16(x), grey32(x), greyf(x).`

9.6.1.2 Functions

- **abs(x)**

  Computes the absolute value of `x`.

- **absdiff(x, y)**

  Computes the absolute difference of `x` and `y`. Use this function instead of `abs(sub(x, y))` to avoid problems with unsigned types.

  Note, that support for this function is currently experimental.

- **acos(x)**

  Computes the inverse cosine of `x`.

- **acosh(x)**

  Computes the inverse hyperbolic cosine of `x`.

- **add(x, y)**

  Computes the sum of `x` and `y`. If `x` and `y` are strings, the result is the concatenated string.

  This function is also available as the + operator.

- **all(i)**

  Returns true if all pixels of the image `i` are not 0.

- **and(x, y)**

  Computes the logical *and*: `x && y`. If `x` or `y` is an image, the computation is done pixel wise and the result is of type `IMAGE_GREY_8`, containing only `0` (false) and `1` (true).

  This function is also available as the `&&` operator.

- **any(i)**

  Returns true if at least one pixel of the image `i` is not 0.

- **argmax(arg0, arg1, ..., argN)**

  Takes at least one parameter and returns the index of the argument with the maximal value.

  `argmax(1, 1, 2, 1) == 2,argmin(6, 2, 3, 4) == 1`

  If one argument is an image, the `argmax` **for this image** is computed pixel wise **for this argument**, that is, if the result is the greatest among all arguments, the result index is the index of the image. The result of `argmax()` is an image of type `IMAGE_GREY_8`.

- **argmin(arg0, arg1, ..., argN)**

  see `argmax()`

- **asin(x)**

  Computes the inverse sine of `x`.

- **asinh(x)**

  Computes the inverse hyperbolic sine of `x`.

- **assert(condition, then, error_message)**

  If `condition` evaluates to `true`, then the value of `then` is returned. If the `condition` evaluates to `false`, an exception with the error message `error_message` is thrown.

  ```
  assert(i1 != 0, 42/i1, "You can not divide through zero.")
  ```

- **atan(x)**

  Computes the inverse tangent of `x`.

- **atan2(x, y)**

  Computes the angle of the polar coordinates of the point `(x, y)`.

- **atanh(x)**

  Computes the inverse hyperbolic tangent of `x`.

- **bool(x)**

  Converts `x` to a bool.

  If `x` is a string, this function returns `true`, iff `x` has at least one character.

  If `x` is a scalar, this function returns `true`, iff `x` is not zero.

  If `x` is an image, this function returns an image of type `IMAGE_GREY_8` with the same size as `x` and the pixel value of the result is `1`, when the associated pixel in `x` is not zero and `0` otherwise.

- **ceil(x)**

  Rounds up to the nearest integer which is greater than or equal to `x`.

- **col(i, start=0, end=-1)**

  Extracts columns from the image `i`. The `start` parameter is the index of the first column and `end` is the index of the last column to extract.

  The output is of the same type as `i`.

  `col(i)` returns a copy of `i`. `col(i, 2)` returns the second column of `i`. `col(i, 2, 4)` returns the 2nd,3rd and 4th column of `i`. `col(i, 2, -2)` returns all the second column up to the second last column.

  See also: `row()` and `pixel()`

- **cos(x)**

  Computes the cosine of `x`.

- **cosh(x)**

  Computes the hyperbolic cosine of `x`.

- **depth(i)**

  Returns the depth of the image `i`.

- **div(x, y)**

  Computes the quotient `x/y`. Please note that the result may be `NaN` or `Inf`.

  This function is also available as the `/` operator.

- **equal(x, y)**

  Returns true if `x` is equal to `y`: `x == y`. If `x` or `y` is an image, the computation is done pixel wise and the result is of type `IMAGE_GREY_8`, containing only `0` (false) and `1` (true).

- **exp(x)**

  Computes the base *e* (Euler number) exponential function of `x`.

- **float(x)**

  Converts `x` to a double. This conversion may fail for images and strings.

- **floor(x)**

  Rounds down to the nearest integer which is less than or equal to `x`.

- **format(fmt, val0, ..., valN)**

  Formats the string `fmt` using the values `val0`, ..., `valN`. All substrings `{0}` are replaced with the value `val0`, all substrings `{1}` with the value `val1` and so on. If you want to use an opening or closing bracket `{}`, you must escape it with a second one `{{}}`.

  ```
  format("The image size {0}x{1}x{2}.  The height is {0}px.", i1.w, i1.h, i1.↩
  d)
  ```

- **grey8(i)**

  Converts image `i` to an `IMAGE_GREY_8`.

- **grey16(i)**

  Converts image `i` to an `IMAGE_GREY_16`.

- **grey32(i)**

  Converts image `i` to an `IMAGE_GREY_32`.

- **greyf(i)**

  Converts image `i` to an `IMAGE_GREY_F`.

- **gt(x, y)**

  Returns true if `x` is greater than `y`: `x > y`. If `x` or `y` is an image, the computation is done pixel wise and the result is of type `IMAGE_GREY_8`, containing only `0` (false) and `1` (true).

  This function is also available as the > operator.

- **gteq(x, y)**

  Returns true if `x` is greater than or equal to `y`: `x >= y`. If `x` or `y` is an image, the computation is done pixel wise and the result is of type `IMAGE_GREY_8`, containing only `0` (false) and `1` (true).

  This function is also available as the >= operator.

- **height(i)**

  Returns the height of the image `i`.

- **if(condition, then_value, else_value)**

  If the `condition` evaluates to `true`, then the `then_value` is returned, otherwise the `else_value`.

  If `condition` is an image, the result is an image of type `IMAGE_GREY_F`, containing the value of `then↩ _value` for pixel `i` where `condition[i]` evaluates to `true` and the `else_value` if `condition[i]` evaluates to `false`.

  If the `then_value` or `else_value` parameters are images, the result is an image of type `IMAGE_GRE↩ Y_F`. The pixel values of the result are pixelwise copied either from `then_value` or `else_value` based on the `condition`.

- **int(x)**

  Converts `x` to a long. This conversion may fail for images and strings.

- **length(s)**

  Returns the length of the string `s`.

- **log(x)**

  Computes the logarithm to the base *e* (Euler number) of `x`.

- **lt(x, y)**

  Returns true if `x` is less than `y`: `x < y`. If `x` or `y` is an image, the computation is done pixel wise and the result is of type `IMAGE_GREY_8`, containing only `0` (false) and `1` (true).

  This function is also available as the < operator.

- **lteq(x, y)**

  Returns true if `x` is less than or equal to `y`: `x <= y`. If `x` or `y` is an image, the computation is done pixel wise and the result is of type `IMAGE_GREY_8`, containing only `0` (false) and `1` (true).

  This function is also available as the <= operator.

- **max(x, y)**

  Returns the maximum of `x` and `y`. For images, this is the pointwise maximum.

- **min(x, y)**

  Returns the minimum of `x` and `y`. For images, this is the pointwise minimum.

- **mod(x, y)**

  Computes the modulus `x` modulo `y`: `x % y`.

  This function is also available as the `%` operator.

- **mul(x, y)**

  Computes the product of `x` and `y`: `x * y`.

  You can also multiply a string with a positive integer. The result is the string repeated `x` resp. `y` times.

  This function is also available as the ∗ operator.

- **neg(x)**

  Computes the negative value of `x`: `-x`.

  This function is also available as the unary, right associative − operator.

- **not(x)**

  Computes the boolean negation of `x`: `!bool(x)`.

  This function is also available as the `!` operator.

- **notequal(x, y)**

  Returns true if `x` is not equal to `y`: `x != y`. If `x` or `y` is an image, the computation is done pixel wise and the result is of type `IMAGE_GREY_8`, containing only `0` (false) and `1` (true).

  This function is also available as the `!=` operator.

- **or(x, y)**

  Computes the logical *or* of `x` and `y`: `bool(x) || bool(y)`

  If `x` or `y` is an image, the computation is done pixel wise and the result is of type `IMAGE_GREY_8`, containing only `0` (false) and `1` (true).

  This function is also available as the || operator.

- **pixel(i, x=0, y=0, z=0)**

  Extracts pixel value at coordinate `(x,y,z)` from the image `i`. The coordinate components can be negative, then those are counted from above.

  The output value type is determined by `i`: long for `IMAGE_GREY_8`, `IMAGE_GREY_16`, and `IMAGE_G↩ REY_32`, and float for `IMAGE_GREY_F`.

  `pixel(i)` returns pixel value at (0) from `i`. `pixel(i, 2)` returns pixel value at (2) from `i`. `pixel(i, 2, 4)` returns pixel value at (2,4) from `i`. `pixel(i, 2, -2)` returns pixel value at (2,i.height-3) from `i`.

  See also: `col()` and `row()`

- **pow(x, y)**

  Computes `x` to the power of `y`.

  See also: `sqrt()` and `mul()`

- **round(x)**

  Rounds `x` to the nearest integer.

- **row(i, start=0, end=-1)**

  Extracts rows from the image `i`. The `start` parameter is the index of the first row and `end` is the index of the last row to extract.

  The output is of the same type as `i`.

  `row(i)` returns a copy of `i`. `row(i, 2)` returns the second row of `i`. `row(i, 2, 4)` returns the 2,3 and 4th row of `i`. `row(i, 2, -2)` returns all the second row up to the second last row.

  See also: `col()` and `pixel()`

- **runplugin(path, input0, ..., inputN, param_name0, param_value0, ..., param_nameK, param_valueK)**

  Runs a toolip plugin with the given parameters.

  ```
  runplugin("%ITWMDIR%/bin/arithmetic::Abs", i1, i2, "upcast", true),runplugin("%↩
  ITWMDIR%/bin/arithmetic::Math", i1, "upcast", true, "function", "SIN")
  ```

- **sin(x)**

  Computes the sine of `x`.

- **sinh(x)**

  Computes the hyperbolic sine of `x`.

- **sqrt(x)**

  Computes the square root of `x`.

- **string(x)**

  Converts `x` to a string. This conversion may fail for images.

- **sub(x, y)**

  Subtracts `y` from `x`: `x - y`.

  This function is also available as the binary − operator.

- **tan(x)**

  Computes the tangent of `x`.

- **tanh(x)**

  Computes the hyperbolic tangent of `x`.

- **type(x)**

  Returns the data type of `x` (string, bool, int, float or image).

  This function is also available as `.type` operator: `x.type`.

- **width(x)**

  Returns the width of the image `x`.

  This function is also available as the `.w` operator: `x.w`.

- **x(i)**

  Returns an image with the same size as `i`, but each column contains the column index:

  ```
  0 1 2 3 ...
  0 1 2 3 ...
  0 1 2 3 ...
  0 1 2 3 ...
  ```

See also: `y()`, `z()`

- **y(i)**

  Returns an image with the same size as `i`, but each row contains the row index:

  ```
  0 0 0 0 ...
  1 1 1 1 ...
  2 2 2 2 ...
  3 3 3 3 ...
  ```

  See also: `x(), z()`

- **z(i)**

  Returns an image with the same size as `i`, but each z-layer is a constant 2D image with the z-index.

  See also: `x()`, `y()`

9.6.1.3 Operators

9.6.1.3.1 Arithmetic operators:

- `+` (add) `3+2`
- `-` (sub, neg) `3-2`
- `*` (mul) `3*2`
- `/` (div) `3/2`
- `%` (mod) `3%2`
- `^` (pow) `3^2`
- `-` (neg) `-3`, `3 + -2`

9.6.1.3.2 Comparison operators:

- < (lt) `3 < 2`
- > (gt) `3 > 2`
- <= (lteq) `3 <= 2`
- >= (gteq) `3 >= 2`
- == (equal) `3 == 2`
- `!=` (nequal) `3 != 2`

9.6.1.3.3 Logical operators:

- `&&` (and) `i1 > 2 && i1 < 3`
- || (or) `i1 > 2 || i1 < 3`
- `!` (not) `!(i1 == 2)`

9.6.1.3.4 Helpers:

- `.x` (x) `i1.x`
- `.y` (y) `i1.y`
- `.z` (z) `i1.z`
- `.width`, `.w` (width) `i1.w`, `i1.width`
- `.height`, `.h` (height) `i1.h`, `i1.height`
- `.depth`, `.d` (depth) `i1.d`, `i1.depth`
- `.type` (type) `i1.type`
- `.length` (length) `"Hello".length`
- `.col` (column) `i1.col(0)`
- `.row` (row) `i1.row(0)`

9.6.1.4 Variables

9.6.1.4.1 Input Port Variables

The calculator has four input pins. You are free to use them or not. They can be connected to an image, double, long, bool or string. You can access those ports with the variables `i1`, `i2`, `i3` and `i4`.

9.6.1.4.2 User Defined Variables

You can define custom variables in the toolip dialog window of the calculator plugin. Please make sure, that the variable name starts with a letter. You are only allowed to use letters, digits and underscores inside the name. You cannot use names of pre-defined functions and constants.

Examples:

- valid names: `a`, `b`, `my_var0`, `var1`
- invalid names: `_a`, `10b`, `$a`

9.6.1.4.3 Pre-defined Constants

Some constants are frequently needed in mathematical computations. These are the pre-defined constants:

- `pi` - floating point value 3.14...
- `true` - boolean value *true*
- `false` - boolean value *false*
- `inf` - floating point value *infinity*
- `nan` - floating point value *not-a-number*

See also

plugin RPNC

Keywords:

calculator, arithmetic, calculation, expression, formula, values calculation

9.7 Data Plugins

9.7.1 Plugin Ball

Draw a ball with a given gray value.

A ball is drawn into the output image and the background is set to a specified gray value.

See also

Example of `Ball Plugin` .

- **Inpin 1:** Input image to determine type and size of the output image, must be of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, IMAG↩ E_GREY_D or IMAGE_MONO_BINARY.

- **Outpin 1:** Output image of same type and size as the input image.

Parameters

center_x(long)	Determines the x-coordinate of the center of the ball. It will be used if the parameter *auto* does not initiate the automatic determination of the center coordinate in x-direction. By default it is set to 0.
center_y(long)	Determines the y-coordinate of the center of the ball. It will be used if the parameter *auto* does not initiate the automatic determination of the center coordinate in y-direction. By default it is set to 0.
center_z(long)	Determines the z-coordinate of the center of the ball. It will be used if the parameter *auto* does not initiate the automatic determination of the center coordinate in z-direction. By default it is set to 0.
auto(string)	Determines whether the center of the image is used as the center of the ball in any dimension: If *auto* is set to *X*, *center_x* will be in the middle of the image width, if *auto* is set to *Y*, *center_y* will be in the middle of the image height, and if *auto* is set to *Z*, *center_z* will be in the middle of the image depth. Similar behaviour for *auto* set to *XY*, *XZ*, *YZ* or *XYZ*. By default it is set to *OFF*.
radius(long)	Determines the radius of the ball. By default it is set to 50.
bg_↩ value(double)	Determines the gray values in the background (outside the ball) of the output image. By default it is set to 0.0.
fg_↩ value(double)	Determines the gray values in the foreground (inside the ball) of the output image. By default it is set to 1.0.

Keywords:

balls, spheres, draw, fill, mask, cut out

9.7.2 Plugin Box

Plugin draws a 2d or 3d box and optionally fills background and foreground with given values.

A 2d box or a 3d box is drawn filled or not into the output image. On a 3d image a 2d box with z > 1 would be drawn slice-wise. Whereas 3d box is drawn as a cuboid.

An example of the `Box Plugin` .

- **Inpin 1:** Input image to determine type and size of the output image. Must be of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32 or IMAGE_GREY_D.

- **Inpin 2:** Optional Second input image with 4 columns and n rows encoding the 2d coordinates of n boxes to be drawn or with 6 columns and n rows encoding the 3d coordinates of n 2d boxes drawn slice-wise or n cuboids denpend on the parameter *box_type*, see below. If the second input is used offset parameters for x and y or by 3d coordinates for x, y and z are ignored.

The box coordinates are specified by four or six entries in each row. These entries define the coordinates of left, right, top and bottom or left, right, top, bottom, near and far corners of each box. It must be of type IMAGE_G←
REY_F.

- **Outpin 1:** Output image of same type and size as the input image.

Note

If the box lies partially outside of the image it will be restricted to the image area. Boxes that lie completely outside of the image are ignored.

Parameters

offset_x(long)	Determines the starting point of the box in x-direction. By default it is set to 0.
offset_y(long)	Determines the starting point of the box in y-direction. By default it is set to 0.
offset_z(long)	Determines the starting point of the box in z-direction. By default it is set to 0.
size_x(long)	Determines the end point of the box in x-direction. If *size_x* < 0, the box ends at the image boundary. By default it is set to -1.
size_y(long)	Determines the end point of the box in y-direction. If *size_y* < 0, the box ends at the image boundary. By default it is set to -1.
size_z(long)	Determines the end point of the box in z-direction. If *size_z* < 0, the box ends at the image boundary. By default it is set to -1.
fillBox(bool)	Determines whether the box is filled with *fg_value* or not. By default it is set to *false*.
fill← Background(bool)	Determines whether the original data (outside the box or in- and outside of the rectangle) remains unchanged or is set to *bg_value*. By default it is set to false.
bg_← value(double)	Specifies the gray value for the background of the output image. By default it is set to 0.
fg_← value(double)	Specifies the gray value for the drawn box or rectangle in the output image. By default it is set to 1.
line_width(long)	Determines the thickness of the box lines or planes that are drawn. On even value of line_← width the outside enlargement is set wider by one pixel then the inside enlargement. By default it is set to 1.
box_type(std← ::string)	Determines the drawn object type. Available values are '2D-BOX' and '3D-BOX', which draw a rectangle or a cuboid into the output image. Note, on a 3d image a 2d box with z $>$ 1 would be drawn slice-wise. Whereas 3d box is drawn as a cuboid. By default it is set to '2D-BOX'.

Keywords:

boxes, rectangle, cuboid, draw, fill, mask, cut out

9.7.3 Plugin Circle

Draw circles.

Draw circles. If no second input is provided, the plugin draws the circle described by the given parameters into the output image. By providing a second input image as a table the list of circles is drawn, see description below. Plugin uses the simple Bresenham algorithm.

- **Inpin 1:** Input image to determine type and size of the output image. It must be of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32 or IMAGE_GREY_D.
- **Inpin 2:** Optional second input image encoding the coordinates and radii of the circles to be drawn. Each row of this image represents one circle. It must have at least 2 columns for the center of a circle, if it has a third column then this column will be used as the radius of the circles. Similarily if it has a 4th column this column will be used as the gray value of the circles. Other columns of the second image will be ignored. It must be of type IMAGE_GREY_F.

- **Outpin 1:** Output image of same type and size as the input image.

Parameters

x_center(long)	Determines the x-coordinate of the center of the circle. By default it is set to 0.
y_center(long)	Determines the y-coordinate of the center of the circle. By default it is set to 0.
radius(long)	Determines the radius of the circle. By default it is set to 10.
color(double)	Determines the color of the circle. By default it is set to 255.0.
fill↩ Background(bool)	Determines whether the circle is drawn into the input image (set to *false*) or in an empty image with same size and type as the input image (set to *true*). By default it is set to *false*.
fillCircle(bool)	Determines whether the circle should be filled with value *color* (set to *true*) or not (set to *false*). By default it is set to *false*.
wrap(bool)	Determines if the circle drawing is performed on a periodic grid. By default it is set to *false*, i.e., no wrapping is performed.

Note

Note that, this plugin works only for 2D images.

See also

`example for plugin Circle`.

Keywords:

circle, draw, fill, mask, cut out

9.7.4 Plugin Constant

Node to create a constant image with given size.

Creates an image of specified size with all pixel values set to a given value.

- **Outpin 1:** Output image of specified size and specified pixel gray value is created, will be of type IMAG↩ E_GREY_F.

Parameters

size_x(long)	Determines the width of the output image. By default it is set to 256.
size_y(long)	Determines the height of the output image. By default it is set to 256.
size_z(long)	Determines the depth of the output image. By default it is set to 1.
constant(double)	Determines the gray value for the output image. By default it is set to 0.0.

type(string)	Determines the type for the output image. Supported values are IMAGE_GREY_8, IMAG↩ E_GREY_16, IMAGE_GREY_32, IMAGE_GREY_F, IMAGE_BINARY_FG. By default it is set to IMAGE_GREY_F.

See also

`example for plugin Constant`.

Keywords:

constant, image, empty image, create image

9.7.5 Plugin HalfPlane

An oriented line segment defines an image region (half plane).

An oriented line segment defines an image region (half plane). This image part is filled with a gray value as specified by the fg_value parameter, the remaining image part is filled with gray value as given by bg_value parameter.

- **Inpin 1:** Input image to determine type and size of the output image. Must be of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Outpin 1:** Output image of same type and size as the input image.

Parameters

ax(long)	It contains the x-coordinate of the first point that determines the line. By default it is set to 0.
ay(long)	It contains the y-coordinate of the first point that determines the line. By default it is set to 0.
bx(long)	It contains the x-coordinate of the second point that determines the line. By default it is set to 10.
by(long)	It contains the y-coordinate of the second point that determines the line. By default it is set to 10.
fg_↩ value(double)	Determines the gray value of the first of the two resulting image parts. By default it is set to 1.
bg_↩ value(double)	Determines the gray value of the second of the two resulting image parts. By default it is set to 0.
verbose(bool)	Determines, whether slope and y-intercept are printed out. By default it is set to *false*.

Keywords:

half plane, fill, region, line

9.7.6 Plugin Line

Node to draw a line segment using simple Bresenham algorithm.

Draws a line segment using simple Bresenham algorithm. An input image of type IMAGE_GREY_F or IMAG↩ E_GREY_8 is read to determine type and size of the output image.
A line segment with value fg_value is drawn through the input image connecting the two given points (ax, ay) and (bx, by). Optionally, the remaining image pixels can be set to bg_value.
A second (optional) input image can be used to draw a set of line segments. Each row of the image data must contain exactly four entries ax, ay, bx, by. Those are denoting the starting point (ax,ay) and the end point (bx,by) for describing a line segment.
Example of the `Line Plugin` .

- **Inpin 1:** Input image to determine type and size of the output image. Must be of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image encoding the coordinates of the lines to be drawn. Must be of type IMAGE_GREY_F.

- **Outpin 1:** Output image of same type and size as the input image.

Note

The input image must be of type IMAGE_GREY_F or IMAGE_GREY_8 and the second (optional) image must be of type IMAGE_GREY_F.

Parameters

ax(long)	It contains the x-coordinate of the starting point of the line. By default it is set to 0.
ay(long)	It contains the y-coordinate of the starting point of the line. By default it is set to 0.
bx(long)	It contains the x-coordinate of the end point of the line. By default it is set to 10.
by(long)	It contains the y-coordinate of the end point of the line. By default it is set to 10.
fg_↩ value(double)	Determines the gray value of the line in the output image. By default it is set to 1.0.
bg_↩ value(double)	Determines the gray value of the background in the output image. By default it is set to 0.0.
fill_↩ background(bool)	Determines whether the line is drawn into the input image (set to *false*) or on a constant image with gray value *bg_value* having the same size and type as the input image (set to *true*).
verbose(bool)	Determines, whether slope and y-intercept are printed onto the screen. By default it is set to *false*.

Keywords:

line, line segment, draw

9.7.7 Plugin Noise

Node to fill the output image with gray values calculated by a range of different noise functions.

The output image is filled with noise of the specified type.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8. Determines type and size of the output image.
- **Inpin 2:** (optional) Optional second input image of type IMAGE_GREY_F and of the same width as the first input image. Needs to have as many rows as the chosen noise type has parameters. It sets the noise parameters independently for every column of the output image. Image must be of type IMAGE_GREY_F.

- **Outpin 1:** Output image of same type and size as the input image.

Provides the following noise types:
UNIFORM

$$p(z) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq z \leq b \\ 0 & \text{otherwise} \end{cases}$$

GAUSSIAN

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(z-\mu)^2}{2\sigma^2}\right)$$

EXPONENTIAL

$$p(z) = \begin{cases} a\exp(-a_{exp}z) & \text{for } z \geq 0 \\ 0 & \text{for } z < 0 \end{cases}$$

RAYLEIGH

$$p(z) = \begin{cases} \frac{2}{b}(z-a)\exp\left(\frac{(-z-a)^2}{b}\right) & \text{for } z \geq a \\ 0 & \text{for } z < a \end{cases}$$

POISSON countable state space $\{0, 1, \dots\}$ and the rule

$$P(X = k) = \frac{\alpha^k}{k!}\exp(-\alpha), \quad \alpha > 0.$$

- **Inpin 1:** Input image to determine type and size of the output image. Must be of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image encoding the parameters independently for every column of the output image. Works only if *type* is set to *UNIFORM*. Each column determines a parameter set which consists of minimum and maximum Must be of type IMAGE_GREY_F.
- **Outpin 1:** Output image of same type and size as the input image.

Parameters

type(string)	Determines the type of noise (*UNIFORM*, *GAUSSIAN*, *EXPONENTIAL*, *RAYLEIGH*, *POI↩ SSON*). By default it is set to *UNIFORM*.
minimum(double)	Determines the minimum gray value for the noise. Works only if *type* is *UNIFORM*. By default it is set to 0.0.
maximum(double)	Determines the maximum gray value for the noise. Works only if *type* is *UNIFORM*. By default it is set to 1.0.
mu(double)	Only needed if *type* is *UNIFORM*. Determines the mean of average value. By default it is set to 0.5.
sigma(double)	Determines the variance. Only needed if *type* is *UNIFORM*. By default it is set to 0.6.
a_exp(double)	Must be positive. Only needed if *type* is *EXPONENTIAL*. By default it is set to 0.5.
a(double)	Only needed if *type* is *RAYLEIGH*. By default it is set to 0.5.
b(double)	Only needed if *type* is *RAYLEIGH*. By default it is set to 0.5.
alpha(double)	Only needed if *type* is *POISSON*. By default it is set to 0.5.
time(long)	Determines whether a time measurement is performed. If *time* > 0, the time that was needed to calculate the result will be printed out on the screen. By default it is set to 0.

Keywords:

noise, fill, random distribution, uniform noise, gaussian noise, exponential noise, rayleigh noise, poisson noise

See also

An example of the `Noise Plugin` .

9.7.8 Plugin PixelValue

Reads a pixel gray value at a specific position in the input image.

Reads a pixel gray value at a specific position in the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** Value of the pixel at the specified location of specified type.

Note

The node can be used to set parameters of other nodes (especially flow control elements) during runtime based on image specific data.

Parameters

pos_x(long)	Determines the x-coordinate of the pixel from where to read the gray value. Negative values are counted from the end of row, i.e. the current position is image width + *pos_x*. By default it is set to 0.
pos_y(long)	Determines the y-coordinate of the pixel from where to read the gray value. Negative values are counted from the end of row, i.e. the current position is height + *pos_y*. By default it is set to 0.
pos_z(long)	Determines the z-coordinate of the pixel from where to read the gray value. Negative values are counted from the end of row, i.e. the current position is depth + *pos_z*. By default it is set to 0.
out_type(string)	Determines the type of the output gray value. Possible types are: *AUTO* (depending on the input type a long value for IMAGE_GREY_8 or a double value for IMAGE_GREY_F is privided as output), *DOUBLE*, *LONG*, *BOOL* (especially useful as input of flow control elements, as they accept only bool values as conditional input), or *STRING*. By default it is set to *AUTO*.
precision(long)	Determines the numerical precision (maximum number of nonzero digits) to be used by the conversion from numerical values to a string. By default it is set to 10.

See also

plugin Point to write a grayvalue at a pixel position

Keywords:

pixel value, position, coordinate, point, greyvalue, grayvalue, pixel sample, readpixel, getpixel

9.7.9 Plugin Point

Set a gray value at a specified position in an image.

Sets a gray value at a specified position in an image.

- **Inpin 1:** Input image to determine type and size of the output image. Must be of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image encoding the coordinate(s) of the pixel(s) to be set. Must be of type IMAGE_GREY_F.
- **Inpin 3:** Optional third input image encoding the gray values to overwrite the gray values at the coordinates given by the second input image, which in this case is mandatory. Must be of type IMAGE_GREY_F.

- **Outpin 1:** Output image of the same type and size as the first input image.

Parameters

px(long)	Determines the x-coordinate of the pixel to be set. By default it is set to 0.
py(long)	Determines the y-coordinate of the pixel to be set. By default it is set to 0.
pz(long)	Determines the z-coordinate of the pixel to be set. By default it is set to 0.
fg_↩ value(double)	Determines the new gray value of the pixel(s) to be set. By default it is set to 1.0.
bg_↩ value(double)	Determines the background gray value in the output image. Only active if *fill_bg* is set to true. By default it is set to 0.0.
fill_bg(bool)	Determines if the background should be filled. By default it is set to true.
verbose(bool)	Determines whether some information is printed out on the screen. By default it is set to *false*.

See also

`Point example graph`.
plugin PixelValue to read a grayvalue from a pixel position

Keywords:

point, position, coordinate, pixel, pixelvalue, writepixel, setpixel, grayvalue, greyvalue

9.7.10 Plugin PositionToValue

Plugin writes the x-, y- or z-index of each pixel coordinate into the output image.

Writes the x-, y- or z-index of each pixel coordinate into the output image.

- **Inpin 1:** Input image to determine size of the output image

- **Outpin 1:** Output image of same size as input image and type IMAGE_GREY_F, where each pixel is set to its position index value.

Parameters

mode(string)	Determines the dimension to be used when setting the output image with coordinate indices. Possible inputs are *X*, *Y*, and *Z*. By default it is set to *X*.

See also

Example of the Plugin `PositionToValue` .

Keywords:

position to value, extract coordinates, fill, get coordinate values

9.7.11 Plugin Spline

Plugin draws a cubic spline into the output image.

Cubic spline(s) are drawn with a specified gray value into the output image. The background can be set to a specified gray value.

- **Inpin 1:** Input image into which the spline is drawn. Must be of type IMAGE_GREY_F or IMAGE_GR↩ EY_8.
- **Inpin 2:** Optional second input image encoding coordinates of the four points that determine the spline(s) to be drawn. Must be of type IMAGE_GREY_F.

- **Outpin 1:** Output image of same type and size as the first input image.

Parameters

ax(long)	Sets the x-coordinate of the first point that determines the spline. By default it is set to 0.
ay(long)	Sets the y-coordinate of the first point that determines the spline. By default it is set to 0.
bx(long)	Sets the x-coordinate of the second point that determines the spline. By default it is set to 2.
by(long)	Sets the y-coordinate of the second point that determines the spline. By default it is set to 5.
cx(long)	Sets the x-coordinate of the third point that determines the spline. By default it is set to 8.
cy(long)	Sets the y-coordinate of the third point that determines the spline. By default it is set to 5.
dx(long)	Sets the x-coordinate of the fourth point that determines the spline. By default it is set to 10.
dy(long)	Sets the y-coordinate of the fourth point that determines the spline. By default it is set to 10.
fg_↩ value(double)	Determines the gray value of the spline(s) in the output image. By default it is set to 1.0.
bg_↩ value(double)	Determines the gray value of the background in the output image in case *fill_background* is set to *true*. By default it is set to 0.0.
fill_↩ background(bool)	Determines whether the spline(s) are drawn into the input image (set to *false*) or in an image filled with *bg_value* with the same size and type as the input image (set to *true*). By default it is set to *false*.
verbose(bool)	Determines whether some calculation information is printed out on the screen. By default it is set to *false*.

Note

plugin uses the De-Casteljau's Divide and Conquer algorithm

Keywords:

spline, cubic spline, draw

9.7.12 Plugin Value

Node to write a given value to the output.

Writes a given value to the output. This node is useful e.g. to generate a parameter for another node or (sub)graph.

- **Outpin 1:** Value of type long, double, bool, string, vector, or map.

Parameters

value	This parameter is of type long, double, bool, string, vector or map and determines the value to be put on the output pin. By default it is set to long(0).

Keywords:

value, create value, bool, long, double, string, vector, map, integer number, floating point number

See also

Example of the node `Value` .

9.8 Features Plugins

9.8.1 Plugin Area

This node calculates an approximated value for the number of labeled pixels. Therefore a bounding box is set up for every label. The approximated number of labeled pixels is the product of the bounding box's height and width.

Calculates an approximated value for the number of labeled pixels. Therefore a bounding box is set up for every label. The approximated number of labeled pixels is the product of the bounding box's height and width.

- **Inpin 1:** Input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times 1$ pixel, where n is the number of labels (= number of rows).
- **Outpin 2:** Value vector with the same information

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

Keywords:

area of bounding box, label image, feature

9.8.2 Plugin AveragePhi

Computes the average over n equally sized circular sections (n pieces of a cake).

Computes the average over n equally sized circular sections (n pieces of a cake).

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (= number of rows) and m is the number of sections (= number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

pieces(long)	Number of equally sized circular sections.
verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.3 Plugin AveragePolar

Calculates the average over n polar sections (m angles and p radii).

Calculates the average over n polar sections (m angles and p radii).

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixels, where n is the number of labels (= number of rows) and m is the number of pieces (= number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

angles(long)	Radius of the resulting pieces.
pieces(long)	Number of angular sections (pieces in a cake).

9.8.4 Plugin BinNbhHist

Node to compute the binary 2x2 configuration histogram.

This plugin computes the binary 2x2 configuration histogram. It can be useful to characterize binary texture data.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixels, where n is the number of labels (= number of rows) and m is the number of pieces (= number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

Keywords:

binnbhhist, binary pattern, histogram

9.8.5 Plugin BinNbhHistPlusXYEntropy

Node to compute the binary 2x2 configuration histogram and a measure of the entropy of the 2d positions of the pixels.

Computes the binary 2x2 configuration histogram. It can be useful to characterize binary texture data. Computes a measure of the entropy of the 2D positions of the pixels obtained from the input image by thresholding above and below an automatically determined value. It can be useful in characterizing random point distributions.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.

- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of pieces (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

Keywords:

binnbhhistplusxyentropy, histogram, binary pattern, entropy

9.8.6 Plugin Border

There is no description available for this plugin.

There is no description available for this plugin.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (= number of rows) and m is the number of pieces (= number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

Keywords:

border, number of pixels of regions border

9.8.7 Plugin Covbin

Calculates the covariance of the pixel gray values in a binary image.

Computes local covariance (min eigenvalue) on a binary image, estimated from the foreground pixels. The output of the eigenvalues is written in ascending order. Useful to detect line segments in the image.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of pieces (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.8 Plugin Entropy

Calculates the entropy of the normalized histogram.

A histogram over the grey values of each label is generated, then normalized and its entropy ist calculated.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of pieces (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.9 Plugin FromGraph

Calculates region features on labels using another ToolIP graph.

Computes features for every region using the given graph. For every region (as defined by the label image) the graph is called and the computed features are collected. The graph must accept two float images, the first one is the image data, the second one is the region mask. The result must be a float image with one row containing the feature values. The feature names can be stored in the image Attributes() map as a vector under the key *feature_names*.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of pieces (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

graph(string)	Defining the path of the graph to be used.
verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.10 Plugin Geometry

Provides some features characterizing the geometry of regions in an image.

This node provides some features characterizing the geometry of the binary input image. These features are:

- Area (number of pixels)
- Major axis length
- Minor axis length
- Orientation (angle in rad)
- Eccentricity
- Border Length

- Fragility
- Regression

See Geometry example graph.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times 8$ pixel, where n is the number of labels (= number of rows)
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.11 Plugin Haralick

Computes the gray level co-occurence matrix and the resulting 2nd order statistics of an image.

This node calculates a vector of 2nd order statistics based on the gray-level co-occurence matrix M. The matrix (glcm) is defined as follows: Element a_{ij} is the number of times that a point with gray level i occurs in a position relative to a point with the gray level j according to the displacement (d_x, d_y). Two modes of operation are supported: Either an explicit displacement operator is given as a parameter, or M is averaged over the 8-neighbourhood of the input pixels. In both cases, the matrix is scaled so that the resulting values represent probabilites (i.e. their sum is 1). The resulting feature set is composed of:

- Angular second momentum
$$\sum_{g=1}^{G}\sum_{g'=1}^{G} a_{gg'}$$
- Contrast
$$\sum_{g=1}^{G}\left(g^2 \underbrace{\sum_{g'=1}^{G}\sum_{g''=1}^{G}}_{|g'-g''|=g} a_{g'g''} \right)$$
- Correlation
$$\frac{1}{\sigma_x \sigma_y}\sum_{g=1}^{G}\sum_{g'=1}^{G}\left(a_{gg'}gg' - \mu_x\mu_y\right)$$
- Variance
$$\sum_{g=1}^{G}\sum_{g'=1}^{G} a_{gg'}(g-\mu)^2$$
- Inverse difference moment
$$\sum_{g=1}^{G}\sum_{g'=1}^{G} \frac{a_{gg'}}{1+(g-g')^2}$$
- Sum average
$$\sum_{g=2}^{2G} g p_{x+y}(g) \qquad (=: F_{\mathrm{SAv}})$$

- Sum variance
$$\sum_{g=2}^{2G}(g-F_{\text{SAv}})^2 p_{x+y}(g)$$
- Sum entropy
$$-\sum_{g=2}^{2G} p_{x+y}(g)\log p_{x+y}(g)$$
- Entropy
$$-\sum_{g=1}^{G}\sum_{g'=1}^{G} a_{gg'}\log a_{gg'}$$
- Difference variance
$$\sum_{g=0}^{G-1}\left(p_{x-y}(g)\left[g-\sum_{g'=0}^{G-1} g' p_{x-y}(g')\right]^2\right)$$
- Difference entropy
$$-\sum_{g=0}^{G-1} p_{x-y}(g)\log p_{x-y}(g)$$
- Information measure of correlation
$$\frac{HXY-HXY1}{\max(HX,HY)}$$
  and
$$\sqrt{1-e^{-2\cdot|HXY2-HXY|}}$$

Abbreviations:

- μ is the mean gray level of the input data
- $p_x(g) := \sum_{g'=1}^{G} a_{gg'}$ (column sums), $p_y(g) := \sum_{g'=1}^{G} a_{g'g}$ (row sums)
- $p_{x+y}(g'') = \underbrace{\sum_{g=1}^{G}\sum_{g'=1}^{G}}_{g+g'=g''} a_{gg'}$ (diagonal sum 1) and $p_{x-y}(g'') = \underbrace{\sum_{g=1}^{G}\sum_{g'=1}^{G}}_{|g-g'|=g''} a_{gg'}$ (diagonal sum 2)
- $\mu_x = \sum_{g=1}^{G} g p_x(g)$ (column sum mean), $\mu_y = \sum_{g=1}^{G} g p_y(g)$ (row sum mean)
- $\sigma_x = \sqrt{\sum_{g=1}^{G} p_x(g)(g-\mu_x)^2}$ (column sum variance), $\sigma_y = \sqrt{\sum_{g=1}^{G} p_y(g)(g-\mu_y)^2}$ (row sum variance)
- Entropy measures: $HXY = -\sum_{g=1}^{G}\sum_{g'=1}^{G} a_{gg'}\log a_{gg'}$, $HXY1 = -\sum_{g=1}^{G}\sum_{g'=1}^{G} a_{gg'}\log(p_x(g)p_y(g'))$, $HXY2 = -\sum_{g=1}^{G}\sum_{g'=1}^{G} p_x(g)p_y(g')\log(p_x(g)p_y(g'))$, $HX = -\sum_{g=1}^{G} p_x(g)\log p_x(g)$, $HY = -\sum_{g=1}^{G} p_y(g)\log p_y(g)$

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

m_dx(long)	Horizontal component of the displacement operator.
m_dy(long)	Vertical component of the displacement operator.
verbose(bool)	Prints function feedback to log area. By default it is set to *false*.
m_glcm_↩ size(long)	Resolution of the gray level co-occurence matrix, e.g. 16 <= *m_glcm_size* <= 256

9.8.12 Plugin HoughEntropy

Node to compute several measures of the entropy of the Hough histogram.

Computes several measures of the entropy of the Hough histogram. If the input image contains points on lines then the Hough histogram contains pronounced peaks and this should be reflected in entropy values computed in the hough space. The entropy values calculated are:

- maximal value devided through sum of foreground pixels
- expectation of normalized density (Reny entropy)
- expectation of logarithmic normalized density (entropy)
- maximal value of histogram

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features (=number of columns).
- **Outpin 2:** Value vector with the same information and meta information for each output feature.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.13 Plugin LBPHist

Node to compute the Local Binary Pattern (LBP) histogram.

This plugin computes the local binary pattern histogram as described by Ojala et al in 1994. Out of the gray value image an 8-digit binary pattern image is created. In each pixel the 8 neighbors are described by their value relative to the pixel itself (1 if it is lower, 0 if it is bigger). Over the 8-digit binary pattern subsequently a histogram is constructed. The node can be useful to characterize texture data.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.14 Plugin Median

Computes the median of grey values for all regions in an image.

Computes the median of grey values for all regions in an image.

Note

If the number of pixels in the region is even then the right (non smaller) value (b) of the middle pair (a <= b) of the sorted values is taken.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times 8$ pixel, where n is the number of labels (= number of rows)
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.15 Plugin NbhSpHist

Computes the histogram over the scalar product of the neighborhood gray values of each pixel.

The sum of the scalar product of each pixel with its 4-neighborhood is calculated and subsequently the histogram over these values built. To counteract bias through extreme values, one can ignore the *m_nNumberOfPixels* highest and lowest values.

- **Inpin 1:** First input image with values only in the range from zero to 255, must be of type IMAGE_GRE↩ Y_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.
m_nNumberOf↩ Pixels(long)	Determines the number of pixels with values above *th_above* and below *th_below*. By default it is set to 30.

9.8.16 Plugin RelnbhHist

This node calculates the neighbourhood relation histogram.

For each region and for all pixels of this region, the difference of gray values to the pixel on its right, bottom and at the bottom right is calculated. If the difference exceeds a threshold (here 5), it is counted as follows: pixel to the right: 0/+1 (small difference/big difference) pixel at the bottom: 0/+2 pixel at the bottom right: 0/+4 These numbers are added up and the counter for the feature with this number (= column in the output image, in the line of the region) is incremented. Finally, the histogram is normalized, that means: all values are divided by the number of pixels of the respective region.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features - in this case 8 (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.17 Plugin RotHistVar

This node calculates the minimal variance of the 2D histograms of the original image and the rotated original image.

Every region is rotated by 10° and the variance between the rotated and the original image is calculated.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F, mandatory.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features - in this case 1 (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.18 Plugin Rough

Calculates the roughness of each region by counting the number of pixels for which the difference between smoothed and original image exeeds some threshold.

For each region, the smoothed image is obtained by simple summing along x and y direction and dividing by the number of foreground pixels. The number of pixels of a region for which the difference between smoothed and original image exceeds a threshold (here 11), is written in the line of this region.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features - in this case 1 (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.19 Plugin SDistMedianqDiff

Calculates the median and q75-q25 quantile difference.

20 output features are calculated at various distances from the object.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features (=number of columns, in this case 20).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.20 Plugin SdistAveVar

This node provides features characterizing the average and variance at different distances from the object center.

Provides features characterizing the average and variance at different distances from the object center. For each object, the center of object and eigenvector corresponding to the smaller eigenvalue is computed. Then it computes the distance between each point of object to the center of object perpendicular to the normalized eigenvector. After that the distance distribution is divided into half of number_of_features equidistance bins of size two (The points of object with distance bigger than number_of_features are not taken into account). Finally for each bin the average and variance is calculated as features. Therefore the features are as follows:

- average of gray value of object with distance between 0 and 2
- variance of gray value of object with distance between 0 and 2
- average of gray value of object with distance between 2 and 4
- variance of gray value of object with distance between 2 and 4
- ...

This plugin has two outputs: First output has number_of_features columns and each column is for one feature. The second output is a valuevector map which has number_of_features keys corresponding to features. Each key has a vector of computed features as "value".

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features - in this case 20 (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

number_of_↩ features(long)	Determines the number of features. By default it is set to 20.
verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.21 Plugin Shadow

This node calculates the mean gray value outside of a region at various positions.

Features 1 to 5 (=columns in output image): For every region, the mean gray value of the pixels outside of this region lying i pixels (1<=i<=5) on the left of a point within this region is calculated. Features 6 to 10: For every region, the mean gray value of the pixels outside of this region lying i pixels (1<=i<=5) on the right of a point within this region is calculated. Features 11 and 12: Here the covariance is calculated.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features - in this case 12 (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.22 Plugin ShiftHistVar

Provides the histogram over the variance of an image and its shifted version.

This node provides the minimal variance over the 2d histogram of the original and the shifted labeled image. If A is an input image and B its shifted version the following covariance is calculated between the two: $Var(A) + Var(B) - \sqrt{4Var(AB)^2 + Var(A) - Var(B)^2})$. This is done in x, and in y-direction, i.e. the features are:

- The minimal variance over the 2d histogram of original and shifted image in x-direction
- The minimal variance over the 2d histogram of original and shifted image in y-direction

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m represents the x and y-direction respectively.
- **Outpin 2:** Value vector with the same information.

Parameters

shift_x(long)	Number of pixels to shift the input image in x-direction.
shift_y(long)	Number of pixels to shift the input image in y-direction.
verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.23 Plugin Statistics

Compute pixel statistics: average, variance, maximum, and minimum of gray value over image pixels.

This plugin provides features with statistical information of the label regions. For each label these features are:

- average of foreground pixel
- variance of foreground pixel

- maximal gray value
- minimal gray value
- sum of foreground pixel (if parameter 'extended' is TRUE)
- sum of squared foreground pixel (if parameter 'extended' is TRUE)
- count of sum of squared foreground pixel (if parameter 'extended' is TRUE)

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, or IMAGE_MONO_BINARY
- **Inpin 2:** Optional second input label image. must be of type IMAGE_GREY_F, IMAGE_GREY_8, I↩ MAGE_GREY_16, IMAGE_GREY_32, or IMAGE_MONO_BINARY. If there is no second input image then the features will be computed for all pixels in the first input image, i.e., the second output image will be the first image filled by ones.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features (=number of columns). Note: if no region is found, the image is empty
- **Outpin 2:** Value vector with four entries with vectors of the same information. subvectors are empty if no region was found.

Parameters

extended(bool)	If set to TRUE, result contains more features, see above. By default it is set to *false*.
verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

Keywords:

grayvalue statistics, pixel statistics, features, average, mean, variance, standard deviation, sdev, minimum, maximum, sum, foreground, count

9.8.24 Plugin ThreePartsRelNbhHist

Computes the histogram over a pixel neighborhood devided in three parts within each region.

Computes the histogram over a pixel neighborhood devided in three parts within each region.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features (=number of columns).
- **Outpin 2:** Value vector with the same information .

9.8.25 Plugin TotalVariation

Computes the total variation within all foreground labels.

Within all labels not equal to zero the total variation over the contained pixels is calculated. The total variation is defined in the continuous case by $\int_\Omega \|\nabla f(x)\|_2 dx$ or in the discrete case by $\sum_{j\in\Omega}(D_x f(j)^2 + D_y f(j)^2 + D_z f(j)^2)^{1/2}$ where D_x denotes the forward differences in x-direction and \$\Omega\$ the image domain.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.

- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features (=number of columns), in this case $m = 1$.
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.26 Plugin VSlope

Calculates a measure of vertical slope within the labels.

For every x-value, the grey value tenedency in vertical direction is checked. That means, for each pixel (top to bottom) it is checked if in relation to 5 pixels above, the gray value has increased or decreased. The number of local increases and decreases is summed up and devided by the total number of pixels taken into account. That means if case of a monotonous vertical shift the value should be highest (values will be in $[-1, 1]$). If the gray values are equally distributed it should be close to zero.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features (=number of columns), in this case $m = 1$.
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.27 Plugin XYEntropy

Computes a measure of the entropy of the 2d positions of the pixels for each label.

Computes a measure of the entropy of the 2d positions of the pixels for each label. The thresholds used are computed by building the histogram over all label-pixels. The gray value with 30 pixels lower or higher are chosen as "below-threshold" and "above-threshold" respectively. The entropy within of the positions of these highest and lowest pixels is then calculated. The plugin can be useful in characterizing random point distributions.

See `XYEntropy example graph`.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (=number of rows) and m is the number of features (=number of columns).
- **Outpin 2:** Value vector with the same information.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.8.28 Plugin XYInteractionHist

Computes the Reney 2D entropy in various directions.

Computes the Reney 2D entropy in various directions.

- **Inpin 1:** First input image of type IMAGE_GREY_F.
- **Inpin 2:** Second input label image.

- **Outpin 1:** Output image of type IMAGE_GREY_F with $n \times m$ pixel, where n is the number of labels (= number of rows) and m is the number of features (= number of columns).
- **Outpin 2:** Value vector, contains the same information as the output image.

Parameters

verbose(bool)	Prints function feedback to log area. By default it is *false*.

9.9 Fast Fourier Transform Plugins

9.9.1 Plugin DCT

Plugin to compute the Discrete Cosine Transform (DCT).

Computes the Discrete Cosine Transform (DCT) of the input image(s) using the FFTW library. The normalization is 1/sqrt(n), where n = number of pixel in the image.

- **Inpin 1:** First input image that defines the real part. Must be of type IMAGE_GREY_F. Must be of type IMAGE_GREY_F.

- **Outpin 1:** First output image of type IMAGE_GREY_F that contains the real part of the DCT.

Note

For 3D images the FFT2D is performed plane by plane.
For 2D images the FFT1D is performed row wise.

Parameters

mode(string)	Defines what kind of DCT is performed. Available values are *DCT1D_FORWARD*, *DC↩ T1D_BACKWARD*, *DCT2D_FORWARD*, *DCT2D_BACKWARD*, *DCT3D_FORWARD*, *D↩ CT3D_BACKWARD*. By default it is set to *DCT2D_FORWARD*.

Keywords:

discrete cosine transform, DCT, FFTW

9.9.2 Plugin FFTW

Plugin to compute the Fast Fourier Transform (FFT).

Computes the Fast Fourier Transform (FFT) of the input image(s) using the FFTW library. The normalization is 1/sqrt(n), where n = number of pixel in the image.

- **Inpin 1:** First input image that defines the real part. Must be of type IMAGE_GREY_F.

- **Inpin 2:** Optional second input image that defines the imaginary part. If no second image is provided, the imaginary part is taken to be 0. Must be of type IMAGE_GREY_F.

- **Outpin 1:** First output image of type IMAGE_GREY_F that contains the real part of the FFT.
- **Outpin 2:** Second output image of type IMAGE_GREY_F that contains the imaginary part of the FFT.

Note

For 3D images the FFT2D is performed plane by plane.
For 2D images the FFT1D is performed row wise.

Parameters

mode(string)	Defines what kind of FFT is performed. Available values are *FFT1D_FORWARD*, *FFT1D↩_BACKWARD*, *FFT2D_FORWARD*, *FFT2D_BACKWARD*, *FFT3D_FORWARD*, *FFT3D↩_BACKWARD*. By default it is set to *FFT2D_FORWARD*.

Keywords:
discrete Fourier transform, DFT, FFT, FFTW

9.10 Filter Plugins

9.10.1 Plugin Average1d

Node to compute the average along a given line segment.

Computes for each pixel in the input image the average value along a given line segment. The length of the filter line and its angle can be set by using the parameters radius and angle.

- **Inpin 1:** First input image of type IMAGE_GREY_F.
- **Inpin 2:** Optional second input image of type IMAGE_GREY_F. Sets the angle of the filter for each pixel individually
- **Inpin 3:** Optional third input image of type IMAGE_GREY_F. Sets the half of the length of the line segment for each pixel individually.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

Note

In case of 3D image data, the input image will be processed plane by plane.

Parameters

radius(long)	Determines the half of the length of the filter line segment along which the average will be computed. Can also be specified by the third input image. In this case, half of length of the filter will be read from the third input image. By default it is set to 10.
angle(long)	Determines the angle of the filter line segment along which the average will be computed in degrees from 0 to 360. Can be set by the second input image. By default it is set to 0.
mode(string)	*STANDARD*, *ANGLEDATA*

9.10.2 Plugin Average3d

Node to perform a 3D averaging filter.

Performs a 3D averaging on the input image. The three dimensions of the filter cube can be set separately.

- **Outpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_G↩ REY_32, or IMAGE_BINARY_FG

- **Outpin 1:** Average image of type IMAGE_GREY_F.

Parameters

step_x(long)	Determines the half of the width of the filter cube in x-direction. By default it is set to 0.
step_y(long)	Determines the half of the height of the filter cube in y-direction. By default it is set to 0.
step_z(long)	Determines the half of the depth of the filter cube in z-direction. By default it is set to 0.
ignore_↩ zero(bool)	Determines whether zeros are taken into account. By default it is set to *false*.

9.10.3 Plugin Average3dMasked

Node to perform a 3D averaging filter.

Performs a 3D averaging on the input image. The three dimensions of the filter cube can be set separately.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Mask image of type IMAGE_GREY_8 or IMAGE_MONO_BINARY

- **Outpin 1:** Average image or Integral image of type IMAGE_GREY_F. depends on parameter result
- **Outpin 2:** Count image of type IMAGE_GREY_F. only present if parameter result=="integral"

Parameters

step_x(long)	Determines the half of the width of the filter cube in x-direction. By default it is set to 0.
step_y(long)	Determines the half of the height of the filter cube in y-direction. By default it is set to 0.
step_z(long)	Determines the half of the depth of the filter cube in z-direction. By default it is set to 0. By default it is set to *false*.
result(string)	result is average image or integral and count image

9.10.4 Plugin BinNbh

Node to compute the local binary 2x2 configurations in the input image.

Computes the local binary 2x2 configurations in the input image. It can be used for texture analysis.

Note

For 3D image data the function is being applied plane by plane.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of same type as the input image.

Parameters

neighbourhood(string)	Determines what kind of neighbourhood has to be used, possible values are 2x2 and 2x2x2. By default it is set to 2x2.

9.10.5 Plugin BoundaryRegion

Node to get only regions of the input image that are touching image boundaries.

Node to get only the boundary regions of the input image. The output image contains the objects at the boundaries for which the corresponding parameter is set to *true*.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of same type as the input image.

Note

A 3D input image will be processed plane by plane.

Parameters

left(bool)	In case it is set to *true*, objects touching the left image border will appear in the output image. By default it is set to *true*.
right(bool)	In case it is set to *true*, objects touching the right image border will appear in the output image. By default it is set to *true*.
top(bool)	In case it is set to *true*, objects touching the top image border will appear in the output image. By default it is set to *true*.
bottom(bool)	In case it is set to *true*, objects touching the bottom image border will appear in the output image. By default it is set to *true*.

9.10.6 Plugin CEShock

Coherence-enhancing shock filter.

Applies the coherence-enhancing shock filter on a 3D input image. Depending on whether a pixel belongs to the influence zone of a maximun or a minimum, this filter applies locally either a dilation or an erosion process in all three dimensions. It creates a sharp shock between two influence zones and produces piece wise constant segmentations. For specific information see [Weick2003]

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

structure_↩ scale(double)	If set to a value larger than 0.0, a presmoothing of the image with a Gaussian of standard deviation *structure_scale* will be performed. By default it is set to 2.0.
integration_↩ scale(double)	If set to a value larger than 0.0, a post-smoothing of the structure tensor with a Gaussian of standard deviation *integration_scale* will be performed. By default it is set to 5.0.
cutoff_↩ precision(double)	Determines the cutoff precision for the convolution steps. By default it is set to 3.0.
time_step_↩ size(double)	Sets the time step for computing the dilation and erosion with the numerical method of Osher and Sethian. For stability reasons, the time step should be chosen not larger than 0.5 for 2D images and not larger than 0.33 for 3D images. By default it is set to 0.33.

number_of_↩ iterations(long)	Sets the number of iterations for which the filter is performed. By default it is set to 5.

Literature:

- [Weick2003] J. Weickert, Coherence-enhancing shock filters. In B. Michaelis, G. Krell (Eds.): Pattern Recognition, Volume 2781 of Lecture Notes in Computer Science, pp. 1-8, 2003.

9.10.7 Plugin Canny

Plugin performs Canny filtering on the input image.

The Canny filtering performs several steps to come to a segmentation of the edges contained in the input image. The aim is on the one hand not to disrupt connected edges and on the other hand to achieve thin edges, ideally one pixel in width.

First, a Sobel filtering is performed, and from this information the orientation of the edges at each pixel is determined. Secondly, all those pixels are removed (i.e. set to zero) that have a neighbouring pixel with a larger value not pointing to the original pixel. From this thinned edges, the segmentation is achieved using two thresholds: All pixels with value above *threshold_high* are directly considered as belonging to an edge. All pixels with value below *threshold_low* are considered as not belonging to an edge. For the pixels with gray values between the two thresholds, it is decisive whether another edge pixel has an orientation pointing to it or not.
Plugin supports two implementations, controlled by parameter *method:*

- NEW: This implementation is the same as the algorithm introduced by J. Canny in his PHD thesis (Canny, 1986)
- OLD: This implementation is also Canny edge detection but with some modifications.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

Note

To suppress the influence of noise, it is a common approach to preprocess the input image by applying a Gauss filter before starting Canny segmentation procedure.

Parameters

threshold_↩ high(double)	Every pixel with a gray value greater than *threshold_high* is considered as belonging to an edge. By default it is set to 10.0.
threshold_↩ low(double)	Every pixel with a gray value less than or equal to *threshold_low* is considered as not belonging to an edge. By default it is set to 0.0.
true_↩ value(double)	Sets the output pixel gray value for edge pixels. By default it is set to 1.0.
method(string)	Determines which implementation of the Canny algorithm will be used (*NEW* or *OLD*, see above). By default it is set to *OLD*.
false_↩ value(double)	Sets the output pixel gray value for non-edge pixels. By default it is set to 0.0.
angle_↩ tolerance(double)	Determines the acceptable bending for an edge to be still considered as connected. It is given in degrees. By default it is set to 30.0.

Literature:

Canny, J., A Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986)

Keywords:

Canny, Sobel, edge detector, edge detection

9.10.8 Plugin Convolve

Compute arbitrary linear filters.

Computes arbitrary linear filters by convolution with the given filter coefficients *filt_cov* or given by the second input image. The convolution with filter in a pixel of image is computed as follow: firstly move the center of filter on the pixel and multiply the filter coefficients by the corresponding image pixel and sum the result and then write this result the same pixel of output image. In other words

$$Conv(x,y) = \sum_{j=0}^{h-1}\sum_{i=0}^{w-1} I(x+i-c_x, y+j-c_y),$$

where $I(x,y)$ is pixel value of input image and w and h are width and height of filter and c_x ans c_y are coordinates of center of filter.

- **Inpin 1:** Input image of type *IMAGE_GREY_8* or *IMAGE_GREY_F*.
- **Inpin 2:** Optional second input image encoding the filter coefficients. The size of this image will be used as *filter_width* and *filter_height*, the pixel grayvalues are used as filter coefficients Note that, before convolution, the filter will be casted to double. It must be of type *IMAGE_GREY_8* or *IMAGE_GREY_F*.
- **Outpin 1:** Output image of same type as the input image.

Parameters

filter_cof(string)	Specifies the filter coefficients. The cofficients of filter has to be separated with blanks. By default it is set to *"1"*. Will not be used when second input filter coefficient image is present.
filter_↩ width(long)	determins the filter's width. By default, it is set to 1. Will not be used when second input filter coefficient image is present.
filter_↩ height(long)	determins the filter's height. By default, it is set to 1. Will not be used when second input filter coefficient image is present.
divisor(double)	Filter divisor. By default it is set to *1.0*.
filter_center_↩ x(long)	Filter center position in x. By default it is set to *0*.
filter_center_↩ y(long)	Filter center position in y. By default it is set to *0*.
boundary_↩ treatment(string)	Indicates how to treat pixels that lie outside of the image borders. Possible options are: • *REFLECTIVE:* outer side of border is reflection of inner side of border. • *PERIODIC:* outer of border the image is periodically repeated. • *VALUE_PAD:* outer of border is filled with the value of parameter *fill_value*. By default it is set to *REFLECTIVE*.
fill_↩ value(double)	pixelvalue when *boundary_treatment* is *VALUE_PAD*. Default: *0.0*.

See also

`convolution example`

Keywords:

linear filter

9.10.9 Plugin Covbin

Node to perform an adaptive anisotropic smoothing by computing the local covariance.

Performs an adaptive anisotropic smoothing by computing the local covariance and splitting up the density into two factors: foreground and background.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

bg_↩ weight(double)	Determines the weight of the background pixels. By default it is set to 0.0.
step(long)	Determines the half size of the filter window. By default it is set to 10.
mode(string)	Determines the operation mode of the smoothing filter: • *COVBIN:* Computes the local covariance on a binary image and returns the smaller eigenvalue. • *COVSMOOTH:* Performs a 1D smoothing in the direction of the largest variance. • *DATAWEIGHT:* Computes the local covariance on a positive image and returns the smaller eigenvalue. • *DATAWEIGHTFAST:* Same as above, but faster. • *PHIMAXSMOOTH:* Performs a 1D smoothing in the direction of biggest contribution. By default this parametre is set to *COVBIN*.
weight_↩ th(double)	The local covariance will only be computed for pixels above *weight_th*. By default it is set to 0.0.
dilation_↩ step(long)	Determines the half size of the rectangular dilation filter applied to the input data, the smoothing (*COVSMOOTH* or *PHIMAXSMOOTH*) will be performed only at those pixels which are non zero after the dilation. By default it is set to 0, i.e. no dilation will be performed.

9.10.10 Plugin Covdet

Node to perform a nonlinear filtering based on the covariance matrix of $(x, f(x,y))or(y, f(x,y))$, where $f(x,y)$ is the image data at position (x,y).

Performs a nonlinear filtering based on the covariance matrix of $(x, f(x,y))or(y, f(x,y))$, where $f(x,y)$ is the image data at position (x,y). For every pixel (x,y) its one dimensional neighbourhood is considered depending on the selected direction (X or Y). These are the pixels $(x-step,y),(x-step+1,y),...,(x,y),(x+1,y),...,(x+step-1,y),(x+step,y)$ or $(x,y-step),(x,y-step+1),...,(x,y),(x,y+1),...,(x,y+step-1),(x,y+step)$.

These pixels $(p_i = (x_i,y_i), i = 1,..,2*step+1)$ and the corresponding image values $f_i == f(x_i,y_i) = f(p_i)$ define a 2D point cloud (x_i, f_i) or (y_i, f_i) (depending on the selected direction X or Y).

Before the computation, the first coordinate of the point cloud is scaled using the lambda parameter: $(x_i, f_i) \to (\lambda * x_i, f_i)$ or $(y_i, f_i) \to (\lambda * y_i, f_i)$

In the next step the covariance matrix of the point cloud is computed. The covariance matrix has the diagonal entries aa, bb and the off diagonal entries $ab = ba$ where

$$aa = \lambda * \lambda * (< x * x > - < x > * < x >)$$

$$bb = < f * f > - < f > * < f >$$

$$ab = ba = \lambda * (< x * f > - < x > * < f >)$$

For the Y direction case just replace x with y. <> are the expectation values of the unscaled data, scaling data with lambda scales the entries of the covariance matrix as above.

Once we have the covariance matrix, the eigenvalues, determinant or direction of the eigenvector is computed and written as the result image depending on the *mode* parameter.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Outpin 1:** Output image of type IMAGE_GREY_F

Parameters

lambda(double)	Determines the scaling of the coordinate data (the image gray values are not scaled as only the relative scaling is important and has any effect). Smaller values make the filter more sensitive, bigger values less sensitive. By default it is set to 1.0.
mode(string)	Determines which quantity derived from the covariance matrix is written to the result image. The following modes are supported : • *MINEV:* smaller eigenvalue • *MAXEV:* bigger eigevalue • *DET:* determinant • *PHI:* direction of eigenvector (rad) By default it is set to *MINEV*.
step(long)	Determines the size of the filter mask, for every pixel (x,y) the pixels (x-step,y),(y-step+1,y),..., (x,y),..., (x+step-1,x), (x+step,y) are taken into account. If the parameter *direction* is set to *Y*, the corresponding y range is considered. At the image boundary the mask is adjusted (shrinked) so that all pixels lie inside of the image. By default it is set to 10.
direction(string)	Determines the direction in which the filter operation is performed. Supported values are *X* and *Y*. By default it is set to *X*.

9.10.11 Plugin Diffusion

Node to perform the anisotropic diffusion or the topological gradient on the input image.

Performs the anisotropic diffusion or the topological gradient on the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

mode(string)	Defines the mode to be performed. Possible values are: • *AD:* anisotropic diffusion • *TG:* topological gradient
select(string)	Defines for each mode which intermediate result parameter will be selected. Possible values are *lambda*, *c*, and *u*. By default it is set to *u*.
[AD]	Parameters for *AD* (anisotropic diffusion) mode:
ad_steps(long)	Number of iterations. By default it is set to 10.
ad_dt(double)	Length of time step. By default it is set to 1.0.
ad_↩ sigma(double)	Variance of Gauss filter. By default it is set to 1.0.
ad_rho(double)	Variance of Gauss filter after derivative. By default it is set to 1.0.
ad_mue(double)	Threshold for Gauss filter. By default it is set to 0.0001.
[TG]	Parameters for *TG* (topological gradient) mode:
tg_steps(long)	Number of iterations. By default it is set to 10.
tg_dt(double)	Length of time step. By default it is set to 1.0.
tg_↩ sigma(double)	Variance of Gauss filter. By default it is set to 1.0.
tg_eps(double)	Accuracy of computation. By default it is set to 0.001.

tg_mue(double)	Threshold for Gauss filter. By default it is set to 0.01.
[CG]	Parameters for conjugate gradient solver:
cg_steps(long)	Number of iterations By default it is set to 50.
cg_eps(double)	Accuracy of computation. By default it is set to 0.0001.

9.10.12 Plugin Distance

Plugin computes the Euclidian distance transformation on the input image.

Plugin computes the Euclidian distance transformation on the input image by using either the *STD* or the *VOR↩ ONOI* method. That it, is assigns to each pixel in the background (pixel value zero) the shortest distance to the foreground (pixel value not zero). Therefore, for input pixels with value zero distance values are present in the output, and for input pixel with a non-zero value the output pixel values are set to zero.

- **Outpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_G↩ REY_32, or IMAGE_BINARY_FG
- **Outpin 1:** Output image of type IMAGE_GREY_F.

Note

Currently the VORONOI mode works only for 2D images.

Parameters

mode(string)	Defines the algorithm to be used for computing the distance transformation. Possible values are *STD* and *VORONOI*. For images containing only very few non zero pixels, the *VORONOI* version is significantly faster than the *STD* version. By default it is set to *STD*.

9.10.13 Plugin FourConnected

Node to produce an image whose components are four-connected.

Produces an image whose components are four connected. It can be useful before performing a labeling operation which assumes four connectivity. In case of a 3D input image, the input image will be processed plane by plane.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Outpin 1:** Output image of type IMAGE_GREY_F.

9.10.14 Plugin Gauss

Gauss filter.

Gaussian filtering using *sigma_x*, *sigma_y*, and *sigma_z* (the latter for 3d images). For all sigmas larger or equal *1.0* it uses the recursive filter algorithm by Young-van Vliet, see [YV95]. Is at least one sigma smaller thab *1.0*, it uses a linear filter algorithm. For sigma smaller or equal *0.0*, the data is just copied; in the other direction the filtering does still happen, if the sigma is larger than *0.0*.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

sigma_x(double)	If set to a value larger than *0.0*, the Gauss filter will be applied in x-direction. By default it is set to *1.0*.
sigma_y(double)	If set to a value larger than *0.0*, the Gauss filter will be applied in y-direction. By default it is set to *1.0*.
sigma_z(double)	If set to a value larger than *0.0*, the Gauss filter will be applied along the z-direction. By default it is set to *1.0*.
run_once(bool)	INTERNAL Indicates whether the node is run only once. By default it is set to *false*.

Literature:

- [YV95] Ian T. Young and Lucas J. van Vliet, Recursive implementation of the Gaussian filter, Signal Processing (Elsevier) 44, pp. 139-151, 1995.

9.10.15 Plugin GaussDerivative

Node to apply the Gauss derivative filter on the input image.

Applies the Gauss derivative filter on the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

sigma-x(double)	If set to a value larger than 0.0, the Gauss derivative filter will be applied along the x-axis. By default it is set to 1.0.
sigma-y(double)	If set to a value larger than 0.0, the Gauss derivative filter will be applied along the y-axis. By default it is set to 1.0.
threshold(double)	Determines the width and height of the filter window. For example, the width of filter is the size of support of $$\frac{1}{\sqrt{2\pi}\sigma_x}e^{-\frac{x^2}{2\sigma_x^2}} - threshold.$$ That means, the values of gaussian above the threshold are determine the width of filter.
verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

Note

The minimum size of filter is 3 (to see the filter size, set the verbose to true).

9.10.16 Plugin IsoNonlinDiffusion

Node to perform isotropic nonlinear diffusion on the input image.

Performs isotropic nonlinear diffusion on the input image, i.e. solves the partial differential equation

$$u_t = \operatorname{div}\Big(g(|\nabla u_\sigma|^2)\nabla u\Big)$$

by Perona and Malik [PM90] and Catt\'e et al. [CLMC92] with additive operator splitting (AOS) [WHV98].

There are many different choices for the diffusivity function g, for example

$$g(s^2) = 1.0/(1.0 + s^2/\lambda^2)$$

(PERONA_MALIK).

If the input image is of type IMAGE_RGB_8 or IMAGE_RGB_8I, the partial diffential equation is

$$u_t^i \;=\; \mathrm{div}\Big(g(\sum_{j=1}^{3}|\nabla u_\sigma|^2)\nabla u^i\Big) \quad ,$$

for i = 1,2,3 (each channel).

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_RGB_8 or IMAGE_RGB_8I.
- **Outpin 1:** Output image of type IMAGE_GREY_F, IMAGE_RGB_8 or IMAGE_RGB_8I.

Parameters

presmoothing_↩ sigma(double)	Standard deviation for Gaussian pre-smoothing kernel. By default it is set to 1.0.
presmoothing_↩ precision(double)	Cutoff precision for Gaussian kernel. By default it is set to 3.0.
diffusivity_↩ function(string)	Choice of diffusivity function g: • *PeronaMalik* • *PeronaMalik2* • *RegularisedTotalVariation* • *Charbonnier* • *Weickert* By default it is set to *PeronaMalik*.
contrast_↩ parameter(double)	Contrast parameter for diffusivity. By default it is set to 1.0.
time_step_↩ size(double)	Time step size for discretisation. By default it is set to 5.0.
number_of_↩ steps(long)	Number of time steps to perform. By default it is set to 2.
process_3d_↩ image(string)	Choice for the processing in 3D. • *VOLUME:* apply a 3D diffusion • *SLICES:* apply the 2d-diffusion plane by plane By default it is set to *VOLUME*.

Literature:

- [PM90] P. Perona and J. Malik, Scale space and edge detection using anisotropic diffusion, IEEE Transactions on Pattern Analysis and Machine Intelligence 12, 629–639, 1990.
- [CLMC92] F. Catt\'e, P.-L. Lions, J.-M. Morel, and T. Coll, Image selective smoothing and edge detection by nonlinear diffusion, SIAM Journal on Numerical Analysis 29 (1), 182-193, Feb. 1992.
- [WHV98] J. Weickert, B. M. ter Haar Romeny, and M. A. Viergever, Efficient and reliable schemes for nonlinear diffusion filtering, IEEE Transactions on Image Processing 7 (3), 398-410, March 1998.

9.10.17 Plugin LBP

Node to compute the local binary pattern of the input image.

Computes the local binary pattern of the input image. The reference pixel value in the middle of a circle is binary coded by the pixel value informations of the points on the circle. If a pixel value of a point on the circle is larger than the reference pixel value, it is coded with 1, otherwise with 0. The resulting binary pattern is assigned to the reference pixel. For 3D images the function is being applied plane by plane.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Outpin 1:** Output image of type IMAGE_GREY_F or IMAGE_GREY_8.

Parameters

radius(long)	Sets the radius around the midpoint. By default it is set to 1.
neighbors(long)	Sets the number of points on the circle. By default it is set to 8.

9.10.18 Plugin Laplace

Laplace filter.

The discrete second-order derivative (Laplacian operator) of input image is estimated by using 3x3 filter mask. This filter highlights regions with rapid density change and therefore is sensitive to noise. This filter often used for edge detection. The Laplacian of an image with intensity values $I(x,y)$ is calculated by the following filter mask:

$$\begin{pmatrix} +1 & +1 & +1 \\ +1 & -8 & +1 \\ +1 & +1 & +1 \end{pmatrix}$$

- **Inpin 1:** Input image to determine type and size of the output image. It must be of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32 or IMAGE_GREY_D.
- **Outpin 1:** Output image of same type and size as the first input image, i.e., the result will be casted to type of input image.

Parameters

boundary_↩ treatment(string)	Determines type of border treatment. Allowed border treatment are: • REFLECTIVE: outer side of border is reflection of inner side of border. • PERIODIC: outer of border the image is periodically repeated. • VALUE_PAD: outer of border is filled with the value of parameter fill_value. By default it is set to REFLECTIVE.

Note

in case of 3D image data, the input image will be processed plane by plane.

See also

example of `LaplaceFilter` .

Keywords:

Laplacian, Laplace, edge detector

9.10.19 Plugin LocalBinaryPatterns

Plugin computes the local binary patterns (LBP) of the input image.

Computes the local binary pattern of the input image. The reference pixel gray value in the middle of a circle is binary coded by the pixel gray value informations of the points on the circle. If a pixel gray value of a point on the circle is larger than the reference pixel gray value, it is coded with 1, otherwise with 0. The resulting binary pattern is assigned to the reference pixel. The LBP can be computed invariant of the rotation of an image. The boundary is treated with reflection.

Note

For 3D images the function is being applied plane by plane.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of type IMAGE_GREY_32, IMAGE_GREY_16 or IMAGE_GREY_8 (depending on parameter *neighbors*).

Parameters

radius(long)	Tthe radius around the midpoint. By default it is set to 1.
neighbors(long)	Number of points on the circle. By default it is set to 8.
rot(bool)	Indicates if LBP should be invariant with respect to rotation. By default it is set to *false*.

Keywords:

local binary pattern, LPB, code

9.10.20 Plugin LowCount

Ordinal transform which assigns rank to pixel, that is, count the number of pixels in the neighborhood whose gray value is less than that of the pixel in the middle.

The plugin computes the ordinal transform, that is, assigning each pixel its rank in its 8-neighbood in 2d, resp. the 26-neihborhood in 3d. Here, the rank is directly the number of neighborhood pixels whose gray value is less than that of the pixel in the middle. The result can be useful as an approximate skeleton from a distance function.

- **Inpin 1:** Input image, must be of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of type IMAGE_GREY_F or IMAGE_GREY_8.

Parameters

shift(double)	Value will be subtracted from the gray value of the middle pixel before performing the comparison. By default it is set to 0.

9.10.21 Plugin Median

Node to perform the Median filter on the input image.

Median filtering of the input image. The Median filter checks whether each pixel is representative of its surroundings inside the rectangular filter window and replaces it with the median value of those values.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as the input image.

Parameters

step_x	(long) Determines the half width of the filter window. By default it is set to 1.
step_y	(long) Determines the half height of the filter window. By default it is set to 1.

method	(string) Determines which implementations should be used. There are the following possibilites: • HUANG: This implementation is introduced by Huang, see [1], and it consists of a moving histogram of filter window. supported only for image type IMAGE_GREY_8. • PERREAULT: This implementation is introduced by Perreault and Hebert, see [2]. supported only for image type IMAGE_GREY_8 • NAIVE: Sorting elements of mask to find the median. • SORTING_NETWORKS: Uses Sorting Networks for fast parallel sorting of arrays of a fixed size. Sorting Networks are e.g. introduced in [3]. supported for image types IMAGE_GREY_8 and GREY_F, only for masksizes smaller or equal to 5x5. For small size of filter window, HUANG method is faster as PERREAULT and for larger size of filter, PERREAULT is faster. By default, the plugin uses HUANG.

Literature:

- [1] Thomas S. Huang, George J. Yang and Gregory Y. Tang. A fast two-dimensional median filtering algorithm. IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 27, no. 1:13-18, 1979
- [2] Simon Perreault and Patrick Hebert. Median filtering in constant time. IEEE Transactions on Image Processing, vol. 16, no. 9:2389-2394, 2007)
- [3] D. E. Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching (Second ed.). Addison–Wesley, 1997. pp. 219–247. Section 5.3.4: Networks for Sorting.

Note

3d images not supported

9.10.22 Plugin Ranking

Rank filter.

Rank filter plugin sorts pixel values in filter window of size (2∗step+1)∗(2∗step+1) and assigns the value at the position rank to the pixel in the output image.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Outpin 1:** Output image of type IMAGE_GREY_F or IMAGE_GREY_8.

Note

In input images of type IMAGE_GREY_F, gray values are assumed to be in the range 0 <= gray value <= 255.

Parameters

step(long)	Determines the half size of the filter window By default it is set to 0.
rank(long)	Determines the position in the ordered list of the pixel values. There are exactly (2∗step+1)∗(2∗step+1) positions.

9.10.23 Plugin RidgeDetector

Node to perform ridge detection by using the Gaussian equation and its derivates.

Performs ridge detection by using the Gaussian equation and its derivates.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** First output image of type IMAGE_GREY_F, containing the result of the ridge detection.
- **Outpin 2:** Second output image containing the original input image with Gaussian smoothing applied.
- **Outpin 3:** Third output image of type IMAGE_GREY_F containing the calculated maximum eigenvalues.
- **Outpin 4:** Fourth output image of type IMAGE_GREY_F containing the calculated minimum eigenvalues.

Parameters

sigma(double)	Determines the standard deviation used in the Gaussian equation. By default it is set to 1.0.
gauss_↩ threshold(double)	Determines the size of the Gaussian filter. By default it is set to 1.0 / 10000.0.
thresh-old(double)	Determines the threshold for the maximum eigenvalue. By default it is set to 1.0 / 10000.0.
thresh-old2(double)	Determines the threshold for the first eigenvector. If it is omitted, 2∗*threshold* will be used. By default it is set to 1.0 / 10000.0.

9.10.24 Plugin RobertsCross

Node to perform the Roberts cross filter on the input image.

Performs the Roberts cross filter for edge detection on the input image. The Roberts cross filter performs a 2D spatial gradient computation on the input image and emphasizes therefore regions of high spatial frequency corresponding to edges. Note that, this filter uses mirroring policy for the pixels on the boundary of the image.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

direction(string)	Defines along which axis the filter operation will be performed. • *X:* along the x-axis • *Y:* along the y-axis • *BOTH:* the output is the sum of the absolute values of the filter results along both axes. By default it is set to *BOTH*.
pseudo-filter(bool)	If the Roberts cross filter in *BOTH* directions is needed, one can approximate it by $$\lvert(i,j)-(i+1,j+1)\rvert+\lvert(i+1,j)+(i,j+1)\rvert.$$ This so-called Roberts cross pseudo filter is applied when this parameter is set to *true*. In this case, direction is automatically being used as *BOTH*. By default it is set to *false*.

9.10.25 Plugin Sobel

Sobel filter for edge detection.

Performs the Sobel filter for edge detection. The Sobel filter performs a 2D spatial gradient computation on the input image and thus emphasizes regions of high spatial frequency that correspond to edges. It is used to find the approximate absolute gradient magnitude at each point in the input image.

Note

supports 2d images only

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

direction(string)	Defines in which direction the filter operation is performed. • *X:* Along the x-axis • *Y:* Along the y-axis • *X_UNIT:* Normalized along the x-axis • *Y_UNIT:* Normalized along the y-axis • *BOTH:* The output is the amplitude (length) of the gradient vector • *DIAG_POS:* Direction of filtering corresponds to a line with slope 1. • *DIAG_NEG:* Direction of filtering corresponds to a line with slope -1. • *DIAG_BOTH:* The output is the sum of the absolute values of the filter results in both diagonal directions. By default it is set to *BOTH*.

Literature:

Gonzalez/Woods, 2nd International Edition.

9.10.26 Plugin StructureTensor2D

Node to calculate the structure tensor of a 2D image.

This node calculates the structure tensor (Foerstern and Guelch, 1987) for a 2d image. For an image f, the structure tensor is defined as

$$J_\rho(\nabla u) \;=:\; K_\rho * (\nabla u \nabla u^T)$$

with a Gaussian kernel K_ρ of standard deviation ρ and the pre-smoothed version $u = K_\sigma * f$ of the input image.

Since the structure tensor is a symmetric 2x2-matrix, it is sufficient to return three images containing the three upper diagonal components: component st11, st12, and st22.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Component st11 of the structure tensor of type IMAGE_GREY_F.
- **Outpin 2:** Component st12 of the structure tensor of type IMAGE_GREY_F.
- **Outpin 3:** Component st22 of the structure tensor of type IMAGE_GREY_F.

Parameters

noise_scale_↩ sigma(double)	Sets the standard deviation for Gaussian pre-smoothing kernel.
difference_↩ scheme(string)	Defines the derivative approximation scheme. By default it is set to *SOBEL*.
integration_↩ scale_↩ rho(double)	Sets the standard deviation for Gaussian post-smoothing kernel.
convolution_↩ precision(double)	Cutoff precision for Gaussian kernels.

See also

StructureTensorEVD2D

9.10.27 Plugin StructureTensorEVD2D

Node to calculate the eigenvector decomposition of the structure tensor of an image.

Computes the eigenvector decomposition of a structure tensor of a 2D image. Useful to detect directions. The output images contain the rotation angle for the eigensystem, the dominant eigenvalues, and the eigenvalues with smaller modulus.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Rotation angle for the eigensystem as image of type IMAGE_GREY_F.
- **Outpin 2:** Dominant eigenvalues as image of type IMAGE_GREY_F.
- **Outpin 3:** Eigenvalues with smaller modulus as image of type IMAGE_GREY_F.

Parameters

noise_scale_↩ sigma(double)	Sets the standard deviation for Gaussian pre-smoothing kernel.
difference_↩ scheme(string)	Defines the derivative approximation scheme. By default it is set to *SOBEL*.
integration_↩ scale_↩ rho(double)	Sets the standard deviation for Gaussian post-smoothing kernel.
convolution_↩ precision(double)	Cutoff precision for Gaussian kernels.

See also

StructureTensor2D

9.10.28 Plugin StructureTensorEVD3D

Plugin calculates the eigenvector decomposition of the structure tensor of an image.

Computes the eigenvector decomposition of a structure tensor of a 3D image. Useful to detect directions.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** largest smallest eigenvalues as image of type IMAGE_GREY_F.

- **Outpin 2:** median eigenvalues as image of type IMAGE_GREY_F.
- **Outpin 3:** smallest eigenvalues as image of type IMAGE_GREY_F.
- **Outpin 4:** eigenvector entry x belonging to the dominant eigenvalues as image of type IMAGE_GREY_F.
- **Outpin 5:** eigenvector entry y belonging to the dominant eigenvalues as image of type IMAGE_GREY_F.
- **Outpin 6:** eigenvector entry z belonging to the dominant eigenvalues as image of type IMAGE_GREY_F.

Parameters

noise_scale_↩ sigma(double)	Sets the standard deviation for Gaussian pre-smoothing kernel.
difference_↩ scheme(string)	Defines the derivative approximation scheme. By default it is set to *SOBEL*.
integration_↩ scale_↩ rho(double)	Sets the standard deviation for Gaussian post-smoothing kernel.
convolution_↩ precision(double)	Cutoff precision for Gaussian kernels.

Note

parameter difference_scheme is not used

Keywords:

structure tensor, Gauss, EVD, eigenvalue decomposition, eigenvector

See also

StructureTensor3D

9.10.29 Plugin Variance

Node to compute the variance of the input image.

Computes the variance of the input image by using an averaging window. The half width of the averaging window can be arbitrarily set.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

step_x(long)	Determines the half width of the averaging window. By default it is set to 10.
step_y(long)	Determines the half height of the averaging window. By default it is set to 10.
step_z(long)	Determines the half depth of the averaging window. By default it is set to 10.

9.10.30 Plugin ceShock_IsoNonlinDiff

Node to apply a combination of the coherence-enhancing shock filter and the isotropic nonlinear diffusion on the input image.

Applies a combination of the coherence-enhancing shock filter and the isotropic nonlinear diffusion on the input image. Depending on a user-defined threshold, the method decides whether to apply a shock step or a diffusion step in each single pixel. The threshold will be compared with either the eigenvalues of the structure tensor or the values computed by a Sobel mask.

See also

CEShock and IsoNonlinDiffusion.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

Preci-sion(double)	Determines the cutoff precision for the convolution steps. By default it is set to 3.0.
Thresh-old(double)	Used to decide whether a shock or a diffusion step is applied. By default it is set to 0.1.
Time-step(double)	Sets the time step for computing the dilation and erosion with the numerical method of Osher and Sethian. The value should be chosen from the range [0.0, 0.5]. By default it is set to 0.5.
Difference_↩ Scheme(string)	Describes the scheme used for creating the structure tensor. Possible values are *CENTRAL↩ _DIFFERENCE*, *SOBEL*, and *KUMAR*. By default it is set to *SOBEL*.
contrast_↩ parameter(double)	Sets the contrast parameter for diffusivity. By default it is set to 1.0
decision-criterion(string)	Describes if the threshold is compared with the *Sobel* mask or with the *Eigenvalues*. By default it is set to *Eigenvalues*.
Number_of_↩ iterations(long)	Bives the number of iterations for which the filter is performed. By default it is set to 10.
diffusivity_↩ function(string)	Choice for the diffusivity function used by the isotropic nonlinear diffusion. Can be of type *PeronaMalik*, *PeronaMalik2*, *RegularisedTotalVariation*, *Charbonnier*, and *Weickert*. By default it is set to *PeronaMalik*.
Structure-scale(double)	If set to a value larger than 0.0, a presmoothing of the image with a Gaussian of standard deviation *Structure-scale* will be performed. By default it is set to 2.0.
Integration-scale(double)	If this parameter is set to a value larger than 0.0, a post-smoothing of the structure tensor with a Gaussian of standard deviation *Integration-scale* will be performed. By default it is set to 5.0.

9.11 Handling Plugins

9.11.1 Plugin GraphFromFile

Run another (sub)graph within the current graph.

Run another (sub)graph within the current graph.

- **Inpin 1:** Input image of type and size depending on the specific input needed by *graph*.

- **Outpin 1:** Output image of *graph*. Type and size are defined by *graph*.

Parameters

graph(string)	Path of the graph to be applied, can include environment variables. By default, it is set to NULL.
verbose(bool)	

Note

in the filename environment variables can be used. The variable must be enclosed in percent signs, e.g. "%↩ TMP%/graph.tlp". If the environment variable is not set, the replacement is empty.

See also

GraphOnLabel, *GraphOnZone*, *GraphOnSlice*

Keywords:

run, graph

9.11.2 Plugin GraphOnLabel

Run a graph on the input image only on specified regions or objects.

For every label from label image, a subimage is extracted from the data image. The given graph is applied to every subimage, the resulting image constitutes of the result images for every label combined according to the label image. Note, that all parameters of the plugin are passed to the specified graph and, so that the graph can use them.

- **Inpin 1:** First input image of type IMAGE_GREY_F.
- **Inpin 2:** Second label input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F and size as the label image.

Parameters

graph(string)	Path of the graph to be applied, can include environment variables. By default,it is set to NULL.
verbose(bool)	send some information to standard output. By default, it is to false.

Note

in the filename environment variables can be used. The variable must be enclosed in percent signs, e.g. "%↩
TMP%/graph.tlp". If the environment variable is not set, the replacement is empty.

See also

GraphOnZone, *GraphOnSlice*

Keywords:

graph, labels, regions

9.11.3 Plugin GraphOnSlice

Run a graph on image slices of 3d images.

Run a graph on image slices of 3d images. The plugin cuts out each 2d slice in the given direction, processes it, and the result will be assembled back into a 3d image.

- **Inpin 1:** 3d input image to be sliced, type IMAGE_MONO_BINARY, IMAGE_GREY_8, IMAGE_GR↩
EY_16, IMAGE_GREY_32, or IMAGE_GREY_F.
- **Inpin 2:** optional second input image of same size as input
- **Inpin 3:** optional third input image of same size as input
- **Inpin 4:** optional fourth input image of same size as input

- **Outpin 1:** optional output image of any type, and same size as input

- **Outpin 2:** optional output image of any type, and same size as input
- **Outpin 3:** optional output image of any type, and same size as input
- **Outpin 4:** optional output image of any type, and same size as input

Parameters

slicetype(string)	one of "XY", "XZ", and "YZ".
graph(string)	Specifies the path to the graph to be applied, can include environment variables. By default,it is set to NULL.
verbose(bool)	send some information to standard output. By default, it is to false.

Note

in the filename environment variables can be used. The variable must be enclosed in percent signs, e.g. "%↩ TMP%/graph.tlp". If the environment variable is not set, the replacement is empty.
no 2d images supported

See also

GraphOnLabel, *GraphOnZone*, *GraphFromFile*

Keywords:
graph, zone, region, slice

9.11.4 Plugin GraphOnZone

Run a graph on image regions defined by the zone image.

Every row of the second input image specifies a rectangular region (zone) in the input image. The region coordinates must be specified per row as pixel gray values in the order left, right, top, bottom. In 3d mode, there must be front nack coordinates as well. For every zone, a subimage is extracted from the first input image and the specified graph is applied to every subimage. The resulting output image of this specified graph constitutes of one row and output_count columns and this row would be copied to the corresponding row of output image of plugin. Note, that all parameters of the plugin are passed to the specified graph and, so that the graph can use them.

- **Inpin 1:** First input image of type IMAGE_GREY_F.
- **Inpin 2:** Second input image (zone image) of type IMAGE_GREY_F. In 2d mode, it must have width of 4, in 3d mode 6.
- **Inpin 3:** Optional input image that would be passed to specified graph as second input image of this graph.
- **Inpin 4:** Optional input image that would be passed to specified graph as third input image of this graph.

- **Outpin 1:** Output image of type IMAGE_GREY_F, the height equals the height of the zone image.

Parameters

graph(string)	Specifies the path to the graph to be applied, can include environment variables. By default, it is set to string "NULL".
output_↩ count(long)	Number of columns the output image is expected to have. By default, it is set to 1.
is3d(bool)	3d image mode. By default, it is set to false.
verbose(bool)	send some information to standard output. By default, it is to false.

Note

The filename may use environment variables. The variable name must be enclosed in percent signs, e.↩ g. "%TMP%/graph.tlp", for 'graph.tlp' located in the temp folder. If the environment variable is not set, the replacement is empty.
in 2d mode, only the first slice of an 3d image is processed

See also

GraphOnLabel, *GraphOnSlice*

Keywords:

graph, zone, region

9.11.5 Plugin Loop

Node to run a graph on (many) input image(s).

Every input image as specified by the image list is read and the specified graph is applied to it. The result image is saved using the same name as the input image but with a different extension.

Parameters

input_↩ files(string)	Specifies the path of a text file containg the list of image paths to the images that will be processed. By default it is set to *NULL*.
extension(string)	Determines the extension of the result image: "name of result image" = "name of input image" + "." + *extension*. By default it is set to *pgm*.
graph(string)	Specifies the path to the graph to be run on every tile. It can contain environment variables in the "%NAME%" syntax. By default it is set to *NULL*.
verbose(string)	Prints function feedback to log area. By default it is set to *OFF*.

9.11.6 Plugin LoopAndTile

Node to run a graph on tiles of (many) input image(s).

Every input image as specified by the image list is read and tiled. On every tile the specified graph is run. The results are put together in one image to produce the result image.

- **Inpin 1:** Input image of type IMAGE_GREY_8.

- **Outpin 1:** Output image of type IMAGE_GREY_8.

Parameters

input_↩ files(string)	Specifies the path to a text file containg the image paths of the images to be processed. By default it is set to *NULL*.
extension(string)	Determines the extension of the name of the result image: "name of result image" = "name of input image" + "." + *extension*. By default it is set to *pgm*.
graph(string)	Specifies the path to the graph to be run on every tile. It can contain environment variables in the "%NAME%" syntax. By default it is set to *NULL*.
tileX(long)	Determines the number of tiles in x-direction. The size of tiles is determined automatically. By default it is set to 1.
tileY(long)	Determines the number of tiles in y-direction. The size of tiles is determined automatically. By default it is set to 1.
verbose(string)	Prints function feedback to log area. By default it is set to *OFF*.

9.11.7 Plugin ParameterLoop

Run a graph for every parameter value/tuple in a vector.

For every parameter value/tuple as specified in the values vector, the given graph is run. Any output generated is not handled at the moment and will be ignored.

- **Inpin 1:** First input image. All image types defined by the specified graph.
- **Inpin 2:** Second input image.
- **Inpin 3:** Third input image.
- **Inpin 4:** Fourth input image.

Parameters

graph(string)	Specifies the path to the graph to be run. It can contain environment variables in the "%NA↩ ME%" syntax. By default it is set to *NULL*.
names(vector)	Defines the parameter names that are supported by the graph. At the same time it specifies how to form (if any) tuples of values from the provided list. Example: values = 1, 2 ,3 , 4 ,5, 6 names = a,b means that the value list defines a list of value pairs (1,2) (3,4) (5,6) for the parameter pair (a,b).
values(vector)	Specifies the parameter set to be used while running the graph, only numerical values (double and long) are supported. Values forming a tuple should be listed consecutively.
verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

Keywords:

run,graph,parameter tuple

9.11.8 Plugin Paste

Node to paste images vertically and combine them to one image.

Every input image as specified by the image list is pasted vertically to produce the result image.

- **Outpin 1:** Output image of type IMAGE_GREY_8.

Parameters

input_↩ files(string)	Specifies the path to a text file containg the list of absolute file paths to the images to be processed. By default it is set to *NULL*.
verbose(bool)	Prints function feedback to log area. By default it is set to *false*.

9.11.9 Plugin Tile

Node to run a graph on tiles of the input image.

Splits the image into a given number of tiles, calls the given graph on every tile and puts all resulting images together. If a second input image is provided, this will be available on the second input slot of the graph to be run. This second input image will not be tiled and remains the same for all processed tiles.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image, type is determined by *graph*.

Parameters

tileX(long)	Determines the number of tiles in x-direction, the size of tiles is determined automatically. By default it is set to 1.
tileY(long)	Determines the number of tiles in y-direction, the size of tiles is determined automatically. By default it is set to 1.
tile_size_x(long)	Determines the size of tiles in x-direction, the number of tiles is determined automatically. Overrides the value of tileX if *tile_size_x* > 0. By default it is set to -1.
tile_size_y(long)	Determines the size of tiles in y-direction, the number of tiles is determined automatically. Overrides the value of tileX if *tile_size_x* > 0. By default it is set to -1.
graph(string)	Specifies the file path to the graph to be run on every tile. It can contain environment variables in "%NAME%" syntax. By default it is set to *NULL*.

9.11.10 Plugin TileOneInTwoOut

Node to run a graph on tiles of the input image.

Tiles the input image into a given number of tiles, runs the given graph on every tile, puts all resulting images together. Works for graphs producing two output images.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** First output image of type IMAGE_GREY_8.
- **Outpin 2:** Second output image of type IMAGE_GREY_8.

Parameters

tileX()	Determines the number of tiles in x-direction, the size of tiles is determined automatically. By default it is set to 1.
tileY()	Determines the number of tiles in y-direction, the size of tiles is determined automatically. By default it is set to 1.
graph(string)	Specifies the file path to the graph to be run on every tile. It can contain environment variables in "%NAME%" syntax. By default it is set to *NULL*.
verbose(string)	Prints feedback to log area. By default it is set to *OFF*.

9.12 Manipulation Plugins

9.12.1 Plugin Append

Combines two images by appending one to the other.

Appends two images of the same type in a given direction. If the images do not have equal size, the two dimensions of the append border of the smaller image are extended to the maximal size. The additional space is filled with the value given in the parameter *background_value*.

- **Inpin 1:** First input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE↩ _GREY_32, or IMAGE_MONO_BINARY.
- **Inpin 2:** Second input image of the type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IM↩ AGE_GREY_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** Output image with the same type as the first input image if upcast is FALSE, or wider type of both input image types.

Parameters

along(string)	The direction in which the second input image is appended to the first one. Available values are *X*, *Y* or *Z*. By default it is set to *X*.
background_↩ value(double)	Pixel gray value for additional space. By default, it is set to 0.0.
upcast(bool)	if FALSE, second image has to have the same type as the first input image, and result type is the same. if TRUE, the result image type is the wider type of both input image types.

See also

plugins *Replicate*, *ReplicateRow*, *ReplicateColumn*

Keywords:

append, combine, join, attach, side

9.12.2 Plugin Autocrop

Automatically crop an image to the smallest bounding box of its non zero foreground.

Automatic image cropping, i.e. the output image only contains the non zero inner part of the image. Zero-valued rows, columns and slices at the image boundary are being cut off.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as the input image.
- **Outpin 2:** Map with cropping details. The map contains the following entries: offset_x, offset_y, offset_z, size_x, size_y, size_z.

Note

The map in the second output can be used as an input for Extract node.

Keywords:

crop, automatic, extract inner, remove border

9.12.3 Plugin Border

Remove borders from an image.

Crops out a region given by pixel distance to border.

- **Inpin 1:** Input image of type IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, IMAGE_GR↩ EY_F, IMAGE_BINARY_FG, or IMAGE_RGB_8.
- **Outpin 1:** Output image of the same type as the input image.

Parameters

border(long)	pixel distance to border to be removed

Keywords:

crop, extract, border, cut

9.12.4 Plugin CartesianToPolar

Transforms an image from cartesian to polar coordinates.

Transforms an image in cartesian coordinates (x,y) to an image in polar coordinates (r,ϕ) clockwise using the central pixel as origin and the x-direction as starting point. The output image has $width_{out} = \frac{1}{2}\sqrt{width_{in}^2 + height_{in}^2}$ and $heigth_{out} = width_{out} \times 2\pi$. The origin is assumed in the centre of the image, and the gray values are bilinearly interpolated.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, or IMAGE_G↩ REY_32.

- **Outpin 1:** Output image of the same type as the input image.

Note

for 3d images, plugin transforms from cartesian to cylindrical coordinates

See also

The node PolarToCartesian implements the transform in the opposite direction.

9.12.5 Plugin ComposeImage

Composes two images.

Composes two images in 2d. The parameters help to align the images if required. The way the resulting pixel value is determined can be chosen for those image positions, where both input images overlap. Those positions where only one input image is available are filled with the input values of that respective image. The remaining pixels are filled with a given value.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8 or IMAGE_RGB_8.
- **Inpin 2:** Second input image of the same type as the first input image.

- **Outpin 1:** Output image of the same type as the first input image.

Parameters

shift_x(long)	Adds *shift_x* to the automatically determined shift in x-direction. By default it is set to 0.
shift_y(long)	Adds *shift_y* to the automatically determined shift in y-direction. By default it is set to 0.
mode_x(string)	Alignment of the two input images in x-direction. Valid entries are: • *CENTER:* images are centered in width • *LEFT:* images are left-aligned • *RIGHT:* images are right-aligned • *APPEND:* image is appended in x-direction By default it is set to *LEFT*.

mode_y(string)	Alignment of the two input images in y-direction. Valid entries are: • *CENTER:* images are centered in height • *TOP:* images are aligned at the top • *BOTTOM:* images are aligned at the bottom • *APPEND:* image is appended in y-direction By default it is set to *TOP*.
mode_↩ compose(string)	Defines the way the two input images are combined. Valid entries are: • *MEAN:* the output pixel is the mean of the two input pixels • *MAX:* the output pixel is the maximum of the two input pixels • *MIN:* the output pixel is the minimum of the two input pixels • \æ FIRST: the output pixel is the first input pixel • *SECOND:* the output pixel is the second input pixel By default it is set to *MEAN*.
bg_↩ value(double)	Background value, i.e. the value of those pixels that are not filled by the input images. If the input image is of type IMAGE_RGB_8 the background value will be used for all channels. By default it is set to 0.0.
ignore_↩ zero(bool)	defines whether zeros in the input images are ignored or not. This parameter is used in the *FIRST* and *SECOND* mode. By default it is set to *false*.
paralleliza-tion(string)	defines whether a parallelization is being performed. Valid entries are: • PARALLELIZATION WITH OPENMP: In the case of multi core processor and color images, it computes three chanals parallel (each chanal is a thread). • NO PARALLELIZATION: Compute without parallelization. By default it is set to *NO PARALLELIZATION*.

9.12.6 Plugin DeleteDuplicate

Deletes/detects duplicate rows or columns from an image.

Deletes/detects duplicate rows (if *along* is set to 'Y') or columns (if *along* is set to 'X') from an image, such that in the output there is no row resp. column with the same sequence of grayvalues, that is, each row resp. column is unique. A second output gives the occurance count of each row/column: in the first detected unique row/column the number of occurances is written, in each non-unique row/column a zero is written. This can be useful for filtering tables (in form of images).

- **Inpin 1:** Input image of type GREY_F, GREY_8, GREY_16, GREY_32, or MONO_BINARY.

- **Outpin 1:** Image with unique rows/columns.
- **Outpin 2:** Occurance count image. Has either same size as input size in X/Y direction, when *size_count* == SIZE_OF_INPUT', or has size of unique rows/columns when *size_count* == SIZE_OF_RESULT.

Parameters

along(string)	'X' for columns, 'Y' for rows. default: 'X'.
size_↩ count(string)	'SIZE_OF_INPUT' for including non-unique row/column counts (zeroed), 'SIZE_OF_RES↩ ULT' for excluding non-unique rows/columns

Note

Works only for 2D images, for an 3D image, only the first slice is processed

Keywords:

delete, remove, duplicates, rows, cols, columns, make unique

9.12.7 Plugin Deserialize

Redistribute the image data to new, user-defined image dimensions.

Plugin redistributes the input image data to new, user-defined image dimensions. The total pixel count should remain constant. The new image dimension can be provided by parameters or optionally also by a second input image.

- **Inpin 1:** Input image.
- **Inpin 2:** Optional Second input image providing the new image dimensions.

- **Outpin 1:** Output image containing the redistributed data, same image type as input 1.

Parameters

size_x	Size of the output image in x-direction. If the value is 0, the size in x-direction of the input image is kept. By default this is set to 0.
size_y	Size of the output image in y-direction. If the value is 0, the size in y-direction of the input image is kept. By default this is set to 0.
size_z	Size of the output image in z-direction. If the value is 0, the size in z-direction of the input image is kept. By default this is set to 0.

See also

Serialize: transform data into one dimensional image (vector)

Keywords:

serialize, serialise, image size, image dimensions, redistribute data, reshape data

9.12.8 Plugin Expand

Expands an image to its double size in each direction.

Expands an image to twice its width, height an depth by duplicating its rows, columns and slices. In case of a 2d image only width and height are being doubled.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** Output image of the same type as the input image.

See also

Shrink: reduces the size of an image by a factor of two in all directions.

9.12.9 Plugin Extract

Extracts a rectangle or cuboid from an image.

Crops out a rectangle or cuboid given by offset and size.

- **Inpin 1:** Input image of type IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, IMAGE_GR↩ EY_F, IMAGE_BINARY_FG, or IMAGE_RGB_8.
- **Inpin 2:** Map with cropping details. The map can contain the following entries: offset_x, offset_y, offset_z, size_x, size_y, size_z. Any entry found in this input-map overrides the corresponding parameter.

- **Outpin 1:** Output image of the same type as the input image.

Parameters

size_x(long)	width of the extracted region. If *size_x* <0 or *size_x* $>$ the possible width (i.e. input image width - *offset_x*) for extraction, the region ends at the image boundary. By default it is set to -1, which causes the region to end at the image boundary.
size_y(long)	height of the extracted region. If *size_y* <0 or *size_y* $>$ the possible height (i.e. input image height - *offset_y*) for extraction, the region ends at the image boundary. By default it is set to -1, which causes the region to end at the image boundary.
size_z(long)	depth of the extracted region. If *size_z* <0 or *size_z* $>$ the possible depth (i.e. input image depth - *offset_z*) for extraction,, the region ends at the image boundary. By default it is set to -1, which causes the region to end at the$*$ image boundary.
offset_x(long)	starting point of the extracted region in the x-direction. If *offset_x* <0, the offset is counted from the opposite side of the image. By default it is set to 0. If *abs(offset_x)* is larger than the width of the input image, it is automatically corrected to the x-index at the image boundary (i.e. 0 or width-1).
offset_y(long)	starting point of the extracted region in the y-direction. If *offset_y* <0, the offset is counted from the opposite side of the image. By default it is set to 0. If *abs(offset_y)* is larger than the height of the input image, it is automatically corrected to the y-index at the image boundary (i.e. 0 or height-1).
offset_z(long)	starting point of the extracted region in the z-direction. If *offset_z* <0, the offset$*$ is counted from the opposite side of the image. By default it is set to 0. If *abs(offset_z)* is larger than the depth of the input image, it is automatically corrected to the z-index at the image boundary (i.e. 0 or depth-1).
verbose(bool)	verbose mode, print function feedback to log area.

Keywords:

crop, extract, cut

9.12.10 Plugin Flip

Flips/reflects an image in the specified axis directions.

Flips/reflects the input image in the X, Y, and/or Z directions.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_D, IMAGE_GREY_8, IMAGE_GRE↩ Y_16 or IMAGE_GREY_32.

- **Outpin 1:** Output image of the same type as the input image.

Parameters

flip_x(bool)	flips the input image in x-direction. By default it is set to *false*.
flip_y(bool)	flips the input image in y-direction. By default it is set to *false*.
flip_z(bool)	flips the input image in z-direction. By default it is set to *false*.

Keywords:

flip axes, axis, left to right, mirror, reflect

9.12.11 Plugin LabelToColor

Node to change labels to a user-defined color map.

Changes each label in the first input image into the corresponding color in an optional user-defined color map image. The second (color map) image should contain at least as many color values as the first image has labels. If the color map image is omitted a standard color map with six repeating basic colors is used. The parameter *ignore_zero* specifies, whether zeros are taken into account or are ignored. If the map image is a color image every pixel specifies one color, otherwise every consecutive three pixels specify the color as an RGB triple.

- **Inpin 1:** Input label image of type IMAGE_GREY_F, IMAGE_GREY_8.
- **Inpin 2:** Optional (color) map image of type IMAGE_GREY_F, IMAGE_GREY_8 or IMAGE_RGB_8.

- **Outpin 1:** Output label image of type IMAGE_RGB_8.

Parameters

ignore_↩ zero(bool)	defines whether zeros in the label image are ignored (i.e. the corresponding output pixel is set to zero) or not. By default it is set to *false*.

See also

LabelToValue: the corresponding gray-value mapping

9.12.12 Plugin LabelToFeature

Node to map labels of the input image to computed feature values.

Maps the labels of the label image to feature values computed on the labeled regions of the input image. If the label image is omitted, the input image will be used as label image, too. The feature computation is defined by the string *feature_plugin* and the *feature_index*.

Note

Attention: plugin is deprecated and included for backwards compatibility reasons. Works only with "vintage feature" plugins. The functionality now can be created by directly using feature plugins and LabelToValue.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Inpin 2:** Optional label image of type IMAGE_GREY_F.

- **Outpin 1:** Label image of type IMAGE_GREY_F, containing the feature value as label.

Note

Currently only available for 2d images.

Parameters

feature_↩ plugin(string)	path for the feature plugin containing the feature to be computed. By default it is set to *NULL*.
feature_↩ index(long)	index of the computed feature vector. It must be larger or equal to 0. By default it is set to 0.
verbose(bool)	print function feedback to log area.

9.12.13 Plugin LabelToSize

Replaces the label values of the input image by the size values of each labeled region.

Replaces the label values of the input image by the size values of each labeled region.

Note

For 3d images the computation is performed plane-by-plane by default resp. if parameter 'slice-wise' is set to TRUE.

- **Inpin 1:** Label image of type IMAGE_GREY_F.

- **Outpin 1:** Image with assigned sizes of type IMAGE_GREY_F.

Parameters

slice-wise(bool)	optional: for 3d image, do replacement slice-wise if TRUE (default), or volumetric if FALSE

Keywords:

mapping, size, pixel, voxel, count, label

9.12.14 Plugin LabelToValue

Node to change labels to a user-defined gray-value map.

Changes each label in the first input image into the corresponding gray-value in the second, user-defined map image. The second image should contain at least as many gray-values as the first image has labels. The parameter *ignore_zero* specifies, whether zeros are taken into account or are ignored.

- **Inpin 1:** Input label image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE↩ _GREY_32, or IMAGE_MONO_BINARY.
- **Inpin 2:** Input map image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_↩ GREY_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** Output image of the same type as the map image.

Parameters

ignore_↩ zero(bool)	defines whether zeros in the label image are ignored (i.e. the corresponding output pixel is set to zero) or not. By default it is set to *false*.

Note

maximum greyvalue of the label image must not exceed the total size of the map image.

See also

LabelToColor: the corresponding color mapping

9.12.15 Plugin Pad

Enlarge an image by padding it with constant pixels on the image edges.

Pads an image by adding a number of rows at the top and bottom and a number of columns to the left and right of the image. These additional pixels are filled with a constant value.

Note

The parameter *pad_top* here refers to padding in the y-direction below y=0.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** Padded output image of the same type as the input image.

Parameters

pad_left	(long) below x: number of columns added to the left of the image. By default it is set to 0.
pad_right	(long) above x: number of columns added to the right of the image. By default it is set to 0.
pad_top	(long) below y: number of rows added at the top of the image. By default it is set to 0.
pad_bottom	(long) above y: number of rows added at the bottom of the image. By default it is set to 0.
pad_front	(long) below z: number of slices added at the front of the image. By default it is set to 0.
pad_back	(long) above z: number of slices added at the back of the image. By default it is set to 0.
value	(long or double) pixel value for the added rows and columns. By default it is set to 0.0.

Keywords:

pad, embed, enlarge, add border, add edges

9.12.16 Plugin PlaneToVolume

Node to redistribute the image data to new, user-defined image dimensions.

redistributes the input image data to new, user-defined image dimensions. The new image width and height are defined by the second input image. The new depth is computed as $depth_{out} = \frac{width_{in} \times height_{in} \times depth_{in}}{width_{out} \times height_{out}}$. The remaining image pixels are omitted.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Second input image providing the new image dimensions width and height.

- **Outpin 1:** Output image of the same type as the input image, containing the redistributed data.

9.12.17 Plugin PolarToCartesian

Transforms an image from polar to cartesian coordinates.

Transforms an image in polar coordinates (r, ϕ) to an image in cartesian coordinates (x, y). The first row in the input image corresponds to the line starting from the image center in
positive x-direction.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, or IMAGE_G↩ REY_32.

- **Outpin 1:** Output image of the same type as the input image.

Note

for 3d images, plugin transforms from cylindrical to cartesian coordinates

Parameters

ignore_↩ zero(bool)	Defines whether zeros in the input image are ignored (i.e. the corresponding output pixel is set to zero) or not. By default it is set to *false*.

See also

The node CartesianToPolar implements the transform in the opposite direction.

9.12.18 Plugin Reorient

Node to reorient an image using the main orientation of the data in a second input image.

Reorient an image using the main orientation of the data in a second input image. The first (grey valued) input image is rotated about this main direction. The parameter *resize* defines whether the original image size is retained unchanged or the output image size is adjusted by the rotation. The pixel value for those pixels that may not be filled by the rotated image can be set.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Second input image of type IMAGE_GREY_F or IMAGE_GREY_8 used to compute the rotation angle.

- **Outpin 1:** Output image of the same type as the input image.

Note

Currently only implemented for 2d images.

Parameters

bg_↩ value(double/long)	pixel value for background. background are those pixels that are not filled by the rotation. By default it is set to 0.
resize(bool)	resize original image? defines whether the original image size is retained unchanged or the output image size is adjusted by the rotation. By default it is set to *false*.
verbose(bool)	if true, print angle feedback to stdout By default it is set to *false*.

See also

Rotate: implements a basic rotation.

9.12.19 Plugin Replicate

Replicates an image in a given direction.

Replicates an image in a given direction.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, or IMAGE_BINARY_FG.

- **Outpin 1:** Output image of the same type as the input image.

Parameters

along(string)	This parameter can take the values *X*, *Y* or *Z*. It determines the direction in which the input image is replicated. By default it is set to *X*.
count(long)	This parameter specifies the number of replicas of the input image, By default it is set to *1*, i.e. output image equals input image.

9.12.20 Plugin Resample

Resamples an image by using a pixelwise vector shift.

Resamples an image according to shift information given by three input images, one for each direction. The output image contains the (possibly type-casted) pixel values from the first input image (*input*) using the shift vector defined by the second (*shift_x*), third (*shift_y*) and fourth (*shift_z*) image, i.e. output(x,y,z) = input(x+shift_↩ x,y+shift_y,z+shift_z). For shifts outside the input image domain the pixels are set to *bg_value*.

Note

> Input image(s) must be of type IMAGE_GREY_F or IMAGE_GREY_8.
> Float-valued shifts will be rounded down to the nearest integer. Using the parameter interpolation a data interpolation is performed.

- **Inpin 1:** First input image of type IMAGE_GREY_F or IMAGE_GREY_8 containing the data to be resampled.
- **Inpin 2:** Second input image of type IMAGE_GREY_F or IMAGE_GREY_8 containing the displacement in x-direction.
- **Inpin 3:** Third input image of type IMAGE_GREY_F or IMAGE_GREY_8 containing the displacement in y-direction.
- **Inpin 4:** Fourth input image of type IMAGE_GREY_F or IMAGE_GREY_8 containing the displacement in z-direction.
  If this input image is not assigned, the shift value will be set 0.

- **Outpin 1:** Output image of type as the first input image.

Parameters

bg_↩ value(double/long)	This parameter specifies the background value for those pixels with undefined values. By default it is set to 0.
interpola-tion(string)	This parameter determines whhat kind of interpolation (if any) will be used. Possible values are *NONE* and *BILINEAR* (XY-plane). By default ist is set to *NONE*.

9.12.21 Plugin Resize

Resize an 2d image.

Resize an image by given scales in x- and y-direction or optionally with respect to the size of a second input image. The output is computed using a Gaussian kernel.

- **Inpin 1:** First input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional Second input image of type IMAGE_GREY_F or IMAGE_GREY_8 specifying the dimension of the output image.

- **Outpin 1:** Output image of type as the first input image.

Parameters

scale_x(double)	resizing scale in x-direction. By default it is set to 1.
scale_y(double)	resizing scale in y-direction. By default it is set to 1.
keep_ratio(bool)	controls whether, in case of a second input image, the aspect ration of the resized image is kept or not. For ({keep_ratio == true}), resizing is done by the same (smaller) scale in both directions. By default it is set to *false*.

See also

Scale: a centered image resize, keeping the original image dimensions.
Shrink: resize by a factor 2, using a specified method to compute the output values.

Keywords:

resize, shrink, scale, enlarge, expand, 2d

9.12.22 Plugin Rotate

2d clockwise image rotation.

Clockwise planar rotation by a given *angle* in degrees. The parameter *resize* defines whether the original image size is retained unchanged (*resize* := *false*) or the output image size is adjusted by the rotation (*resize* := *true*). The pixel value for those pixels that may not be filled by the rotated image can be set.

- **Inpin 1:** Input image of type *IMAGE_GREY_F* or *IMAGE_GREY_8*.

- **Outpin 1:** Output image of the same type as the input image.

Parameters

an-gle(double/long)	angle in degree used for the clockwise rotation. By default it is set to 0.
bg_↩ value(double/long)	pixel value for those pixels that are not filled by the rotation. By default it is set to 0.
resize(bool)	defines whether the original image size is retained unchanged (*false*) or the output image size is adjusted by the rotation (*true*). By default it is set to *false*.
method(string)	defines the interpolation method being used. Available methods are: • *NEAREST_NEIGHBOR:* Take nearest pixel of the grid, • *BILINEAR:* Fast 2x2-neighborhood interpolation, • *BICUBIC_SPLINES:* More accurate method using splines for 4x4-neighborhood interpolation. By default it is set to *BILINEAR*.
verbose(bool)	print plugin feedback to log area. By default it is set to *false*.

Note

For special angles, where the image is rotated by an integer value *angle* of (n∗180+90) degrees, the plugin rotates the original image in the straightforward way (without interpolation). In all other cases, interpolation methods are used.
Here we shall shortly describe the different interpolation *methods:*

- *NEAREST_NEIGHBOR:* For each Pixel in the result image, the method determines its location in the input image and finds the four surrounding pixels in the input image. For the value of the result pixel it cooses the value of the nearest of the four pixels.

- *BILINEAR:* The method does the same as in Nearest-Neighbor, but it does not simply choose the nearest pixel. Instead it computes the value of the result pixel by linear interpolation, first in x and then in y direction.
- *BICUBIC_SPLINES:* This method chooses a 4x4-neighborhood of the pixel in the input image. It computes the value of the result pixel by using a cubic spline, first in x and then in y direction. The derivative of each pixel is computed by:
  $f'(v_i) = \frac{v_{i+1} - v_{i-1}}{2}$.

Rotating a 3d image results in the 2d rotation of each single slide.

See also

Reorient: implements a special rotation using the main direction of a second input image as rotation angle.

Keywords:

rotate, angle, degree, pi, transform, nearest neighbor, bilinear interpolation, bicubic splines

9.12.23 Plugin Scale

Scale/flip an image.

Scales the image around the center with given scale factors in all three directions. The image size itself will not be changed. Use negative scaling values to flip/mirror image in the given direction. An additional parameter defines the background value, i.e. the value for those pixels not being filled by the scaled image.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type and size as the input image.

Parameters

scale_x(double)	factor used for the scaling in x-direction, negative for flipping. By default it is set to 1.0.
scale_y(double)	factor used for the scaling in y-direction, negative for flipping. By default it is set to 1.0.
scale_z(double)	factor used for the scaling in z-direction, negative for flipping. By default it is set to 1.0.
bg_↩ value(double/long)	background value for those pixels that are not being filled by the scaled image. By default it is set to 0.

Note

image size is not changed
if one scaling factor is zero, the result will be filled with *bg_value*

See also

Resize: a scaled image resize.
Shrink: resize by a factor 2, using a specified method to compute the output values.

Keywords:

scale, upscale, downscale, resize, flip, mirror

9.12.24 Plugin SelectRays

Select image data from an input image by using a label image.

The plugin selects image data from an input image by using a label image. A one-dimensional image subsection in z-direction is called a ray. The selection of rays from the first input image is defined by the label image. Only image rays from the first input image corresponding to non-zero positions of the label image are taken into account. The output image contains these rays arranged in the y-direction, i.e. the output image has the same depth as the first input image, the height corresponds to the number of non-zeros in the label image, and width = 1.

- **Inpin 1:** Input image of type *IMAGE_GREY_F* or *IMAGE_GREY_8*.
- **Inpin 2:** Label image of type *IMAGE_GREY_F* or *IMAGE_GREY_8*.

- **Outpin 1:** Output images of the same type as the input image, but of smaller size.

9.12.25 Plugin Serialize

Serialize an image, that is, arrange the data into a one-dimensional vector.

Serialize an image, that is, arrange the data into a one-dimensional vector. The orientation of the vector is defined by the parameter *direction*.

- **Inpin 1:** Input image

- **Outpin 1:** Output image of same type containing the redistributed data.

Parameters

direction(string)	direction of the resulting vector. Possible values are *X*, *Y*, and *Z*. By default, this is set to *Z*.

See also

Deserialize: transform the data back

Keywords:

deserialize, deserialise, make 1d, one-dimensional, direction, as vector, reshape data

9.12.26 Plugin Shrink

Node to shrink an image by a factor two in all dimensions.

Shrinks an image by a factor two in all dimensions. The parameter *mode* specifies the way the output pixel values are computed:

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as the input image.

Parameters

mode(string)	defines the way the output pixel values are computed: • SKIP: Every second input pixel is simply omitted. • MAX: The output pixel value is the maximal value of a 2x2x2-neighborhood of the original input pixel. • MAXFAST: The output pixel value is the maximal value of a 2x2-neighborhood of the original input pixel. This method also works for 3d-images but is slightly faster than *MAX*, because the maximum is sought only in the x-y-plane. • MIN: The output pixel value is the minimal value of a 2x2x2-neighborhood of the original input pixel. • MINFAST: The output pixel value is the minimal value of a 2x2-neighborhood of the original input pixel. This method also works for 3d-images but is slightly faster than *MIN*, because the minimum is sought only in the x-y-plane. • MEDIAN: A 2x2-neighborhood of the original input pixel is considered and these four pixel values are sorted. The output pixel value is calculated as the average of the two non-extremal values. • AVERAGE: The output pixel is the average value of a 2x2x2-neighbourhood of the original input pixel By default it is set to *SKIP*.

See also

Resize a scaled image resize.
Scale: a centered image resize, keeping the original image dimensions.

9.12.27 Plugin Split

Split an image into two parts along one of the axes.

Splits an image into two parts along the given direction. The size of the resulting images is determined by the parameter *split_ratio*.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** First output image of the same type as the input image.

- **Outpin 2:** Second output image of the same type as the input image.

Parameters

along(string)	the direction in which the input image will be splitted. Available values are *X*, *Y* or *Z*. By default it is set to *X*.
split_↩ ratio(double)	size ratio for the first result image. The size of the first result image along the split direction is given by split_ratio∗original_size. The size of the second result image is such that it contains all the remaining data of the input image. By default, it is set to 0.5

Keywords:

split, divide, half, halve, partition

9.12.28 Plugin SwitchAxes

Plugin to switch image axes.

Switches image axes and transforms the image data accordingly, i.e performs a reflection.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** Output image of the same type as the input image.

Parameters

axes(string)	defines the axes to be switched; valid entries are: • XYZ: nothing is changed • XZY: y- and z-axes are switched, i.e. reflection on y-z bisector • YXZ: x- and y-axes are switched, i.e. reflection on x-y bisector • YZX: firstly x- and y-axes are switched and after that y- and z-axes are switched, i.e. the respective reflections are carried out one after another. Note that the transfomation way is not uniqe. • ZXY: firstly x- and z-axes are switched and after that y- and x-axes are switched, i.e. the respective reflections are carried out one after another. Note that the transfomation way is not uniqe. • ZYX: x- and z-axes are switched, i.e. reflection on x-z bisector • Default value is *XYZ*.

9.12.29 Plugin Translate

Plugin shifts image.

Translates an image with given offsets for all three orientations, the offsets can be negative. For pixels not covered by the translation the pixel value for the background *bg_value* is assigned.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** Output image of the same type as the input image.

Parameters

offset_x(long)	pixel offset for the translation in x-direction. By default it is set to 0.
offset_y(long)	pixel offset for the translation in y-direction. By default it is set to 0.
offset_z(long)	pixel offset for the translation in z-direction. By default it is set to 0.
bg_↩ value(double)	pixel value for those pixels not covered by the translation. By default it is set to 0.
wrap(bool)	translation can be performed as on a periodic lattice. This can be useful for images in polar coordinates. By default it is set to *false* - no wrapping is performed.

Keywords:

shift, move, wrap around, offset

9.12.30 Plugin VolumeToPlane

Node to re-arrange 3d image data into a 2d image, where *n* slices are arranged next to each other and *n* is given by parameter slices_x.

Re-arranges the 3d input image data in such a way that the original *n* slices of input image are appended next to each other. The width of output image is $width_{out} = width_{in} \times slice_x$.

- **Inpin 1:** 3d input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** 2d output image of the same type as the input image, containing the re-arranged data.

Parameters

slices_x(long)	Number of slices in x-direction. By default, it is set to 1 ans all slices sre arranged in y-direction.
background_↩ value(double)	Pixel gray value for additional space. By default, it is set to 0.0,

9.13 Matching Plugins

9.13.1 Plugin FindShape

Node to find image regions with data similar to given template.

This function finds image regions with data that are similar to the given template. The similarity index is computed using the gradient direction based shape model. In the reference image positions with gradient amplitude over a given threshold are found. The direction of the gradient and the coordinates are kept in a list. For every pixel the gradient directions stored in the list are compared with the actual gradient direction.

- **Inpin 1:** Image to search in. Must be of type IMAGE_GREY_F. Mandatory input.
- **Inpin 2:** Template image. Must be of type IMAGE_GREY_F. Mandatory input.
- **Inpin 3:** Optional mask image. Must be of type IMAGE_GREY_F.

- **Outpin 1:** Output image has same size and type as first input image.

Parameters

threshold	threshold parameter for the gradient amplitude. By default it is set to 300, actually the threshold is compared to the gradient amplitude squared and not the gradient amplitude
sp_↩ threshold,scalar	product threshold, scalar product of normalized gradients in template and input image must exceed this threshold for the point to be counted as match, by default it is set to 0.5 which corresponds to angles smaller then 60°
verbose	if selected the number of considered data points (i.e. with gradient > m_threshold) are printed to stdout. By default it is set to false.
parallelization	This parameter is of type string: • "PARALLELIZATION WITH TBB": In the case of multi core processor, it computes parallel. • "NO PARALLELIZATION": Compute without parallelization. By default it is set to "NO PARALLELIZATION".

9.13.2 Plugin Fsdh

Find shape by displacement histogram - class to enhance binary patterns present in the image similar to the given binary template.

This function enhances binary patterns present in the image similar to the given binary template. The similarity index is computed on the basis of maximum value of the displacement histogram. The function works with binary data only.

- **Inpin 1:** First input image of type IMAGE_GREY_F.
- **Inpin 2:** Second input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

step_x	parameter that determines the sampling rate of the input data in the x - direction. By default it is set to 1.
step_y	parameter that determines the sampling rate of the input data in the y - direction. By default it is set to 1.
smooth1	parameter that determines the size of the displacement histogramm first smoothing filter. By default it is set to 0.
smooth2	parameter that determines the size of the displacement histogramm first smoothing filter. By default it is set to 0.
smooth3	parameter that determines the size of the displacement histogramm first smoothing filter. By default it is set to 0.
normalize	if selected the result image is normalized by the number of pixels in the template image. By default it is set to false.

9.13.3 Plugin Match

Class to find image regions with data similar to given template.

This function finds image regions with data that are similar to the given template. The similarity index is computed from the joint gray level histogram using correlation ratio as measure of statistical dependence. The function works best with grey value data.

- **Inpin 1:** Image where regions should be found. Must be of type IMAGE_GREY_F. Input is mandatory.

- **Inpin 2:** Template regions from first input image are matched to. Must be of type IMAGE_GREY_F. Input is mandatory.

- **Outpin 1:** Output image is also of type IMAGE_GREY_F and has the same size as second input image.
- **Outpin 2:** Output image is of type IMAGE_GREY_F with the size (2,1) for translation vector.

Parameters

scale	parameter that determines the sampling rate of the input image. By default it is set to 10.
range_x	parameter that determines the search range of the translations in the x-direction. The actual range is obtained by multiplying with the scale parameter. By default it is set to 5.

range_y	parameter that determines the search range of the translations in the y-direction. The actual range is obtained by multiplying with the scale parameter. By default it is set to 5.
tx_start	parameter that determines the initial image translation in the x-direction. By default it is set to 0.
ty_start	parameter that determines the initial image translation in the y-direction. By default it is set to 0.
print_cr	if selected the correlation ratio is printed to stdout. By default it is set to false.
mode	parameter that determines the similarity measure to use to perform the matching. Possible values are CR (correlation ratio) or MV (determinant of the covariance matrix). By default it is set to CR (correlation ratio).
wrap	boolean switch to turn ON/OFF the wrapping mode in translation. By default it is set to false.
min_point_count	determines the minimum number of points needed to estimate the conditional variance. By default it is set to 20

9.13.4 Plugin TranslateAdd

This plugin translates an image by the vectors specified by non zero pixels of the second input image (template) and add all translated images together can be used to find patterns similar to the template image.

This function enhances binary patterns present in the image similar to the given binary template. The local maxima for possible matches result from summing translates of the original image by the vectors defined by the nonzero pixels of the second input imaage (template) The function works best with binary data.

- **Inpin 1:** Image which will be translated. Must be of type IMAGE_GREY_F. Input is mandatory.
- **Inpin 2:** Image specifying translation. Must be of type IMAGE_GREY_F. Input is mandatory.

- **Outpin 1:** Output image is also of type IMAGE_GREY_F and has the same size as the first input image.

Parameters

normalize	This parameter determines if the result image will be normalized by the number of non zero pixels in the reference image By default it is set to true, i.e. normalization will be applied

9.14 Matrix Plugins

9.14.1 Plugin BinNbhThFlow

Node to compute the binary neighbourhood histograms for for all threshold values between 0 and 255.

This node computes the histograms of binary configurations for all the images resulting by thresholding with values in the range 0..255. It can be used to follow the changes in the derived quantities like the Euler number, specific length, area, volume.. as the binarization threshold changes the only supported neighbourhoods are 2x2 and 2x2x2 the binary configuration for the 2x2 neighbourhood is coded as 1 for (0,0) 2 for (1,0) 4 for (0,1) and 8 for (1,1) whre (i,j) is the relative pixel position with respect to the reference pixel (see below for some examples) 1 0 1 1 0 1 1 1 -> 1 -> 3 -> 10 -> 7 0 0 0 0 0 1 1 0

the binary configuration for the 2x2x2 neighbourhood is coded for the first plane as the 2x2 configuration , the bit pattern for the second plane is shifted by 4 bits.

- **Inpin 1:** Input image of type IMAGE_GREY_8.

- **Outpin 1:** Output image of type IMAGE_GREY_F or IMAGE_GREY_32 and size 16x256 for 2d input, or 256x256 for 3d input.

Parameters

neighbourhood(string)	type of neighbourhood to compute the binary configuration histograms. Valid types are "2x2" (2d images) and "2x2x2" (3d images).
out_type(string)	type of the output image. Possible types are IMAGE_GREY_F (default) and IMAGE_GR↩ EY_32. For large (3d) images it should be set to IMAGE_GREY_32, the histogram counts will be only approximate.

9.14.2 Plugin CovarianceMatrix

Computes the covariance matrix of a given matrix.

Computes the covariance matrix of a given matrix. The covariance matrix is normalized with n the height of the input image, that is, the size in Y direction.

- **Inpin 1:** Input image of type *IMAGE_GREY_F* with the vectors as columns.

- **Outpin 1:** The covariance matrix as *IMAGE_GREY_F* of size m x m, where m is the width of the input image, that is, the size in X direction.
- **Outpin 2:** The mean value for each column of the input image as *IMAGE_GREY_F* of size m x 1, that is, with the width of the input image and height 1.

9.14.3 Plugin Determinant

Plugin calculates the determinant of a matrix with the QR decomposition.

This plugin calculates the determinant of the input image A as matrix. This is done with the QR decomposition. The matrix A must be a square matrix. In case of 3D inputs, only the first plane is processed.

- **Inpin 1:** Input image must be of type IMAGE_GREY_F. Input is mandatory.

- **Outpin 1:** Value of type double

9.14.4 Plugin EVD

Compute Eigenvalues and Eigenvectors.

This plugin computes the Eigenvalues and Eigenvectors on image A as matrix. The resulting Eigenvalues and Eigenvectors are not sorted. The actual computation is perfomed using the TNT/JAMA template library as described on http://math.nist.gov/tnt/documentation.html

Note

The TNT algorithm works only for real-valued matrices.
In the case of a not symmetric matrix, the possible complex eigenvalues are returned as 2x2 blocks, i.e: if the complex eigenvalues are, for example, $u+iv, u-iv, a+ib, a-ib, x$ and y then the eigenvalue matrix looks like:

$$\begin{pmatrix} u & v & 0 & 0 & 0 & 0 \\ -v & u & 0 & 0 & 0 & 0 \\ 0 & 0 & a & b & 0 & 0 \\ 0 & 0 & -b & a & 0 & 0 \\ 0 & 0 & 0 & 0 & x & 0 \\ 0 & 0 & 0 & 0 & 0 & y \end{pmatrix}$$

- **Inpin 1:** Input images must be of type IMAGE_GREY_F or IMAGE_COMPLEX_F. Input is mandatory. Width and height must be the same.

- **Outpin 1:** Eigenvector image, same type as input image.
- **Outpin 2:** Eigenvalue image, same type as input image.

Parameters

mode	"TNT" (only for real-valued matrices), "LAPACK"
tnt_assume_↩ symmetric	input matrices are symmetric: this fastens up EVD in *mode* "TNT"

Keywords:

matrix, Eigenvalue, Eigenvector, EVD, Eigenvalue decomposition

9.14.5 Plugin FeatureImage

Class to compute features of the input image.

Performs the feature computation of the labeled regions of the input image.

- **Inpin 1:** Input (input image) must be of type IMAGE_GREY_F. Input is mandatory.
- **Inpin 2:** Input (label image) must be of type IMAGE_GREY_F. Input is mandatory.

- **Outpin 1:** Output is of type IMAGE_GREY_F and of size 10x10x1

Parameters

feature_plugin	This parameter determines the feature plugin to be applied. By default it is set to NULL.
verbose	If this parameter is set to TRUE, the feature count, the region count and the scale of each region will be displayed. By default it is set to FALSE.

9.14.6 Plugin GaussianMixture

Node to estimate the gaussian mixture or to evaluate it with given parameters.

Estimates the gaussian mixture or evaluates it with given parameters. The background to the algorithms implemented here can be found in section 9.2 of [Bish2006]

This plugin has two modes:

1. if there is only one input image, it performs the Expectation Maximization algorithm on the input image as matrix of data row vectors. It estimates the parameters for the gaussian mixture of the input image. the output image contains the cluster centers (means), the weights for each cluster and the inverse covariance matrix as row vectors.
2. if there are two input images, the first input image contains the data as row vectors, and the second input image contains the parameters for each cluster. the plugin evaluates the gaussian mixture with the given parameters for each vector the result image contains for each vector of the first input image: [weights[k]∗pdf(x,k)] for every cluster k the last value in each row is the value of the Gaussian Mixture

- **Inpin 1:** First input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Second input image of type IMAGE_GREY_F or IMAGE_GREY_8. If the pixel values differ more than 3.0, you should take the LOG of the pixel values, to avoid that (x-mean) gets too big (where x is a data row vector).

- **Outpin 1:** Output image of the same type as the first input image.

Parameters

cluster_↩ count(long)	number of clusters. By default it is set to 2.
iteration(long)	number of iterations. By default it is set to 100.
eps(double)	is added to the diagonal of the covariance matrix. By default it is set to 0.01.
verbose(bool)	print function feedback to log area.
delta(double)	if the difference between the last loglikelihood-function and the new loglikelihood is less than delta, the algorithm will be stopped. By default it is set to 0.001.

Literature:

-[Bish2006] C. M. Bishop: Pattern Recognition and Machine Learning, Springer Science+Business Media, LLC, 2006.

9.14.7 Plugin ICA

This plugin computes an Independent Component Analysis for given data.

Computes an Independent Component Analysis for given data. Each row of the given data is a variable and each column an observation. The routine assumes that the input data has zero-mean.

There are two output images of type IMAGE_GREY_F: The first output image contains the data projected to the independent components. The second output image contains the projection matrix.

- **Inpin 1:** Input images must be of type IMAGE_GREY_F. Input is mandatory.

- **Outpin 1:** Contains data projected to the independent components. It must be of typ *IMAGE_GREY_F*
- **Outpin 2:** Contains projection matrix. It must be of type *IMAGE_GREY_F*

9.14.8 Plugin Invert

Node to invert the input matrix image.

Inverts the input matrix image.

- **Inpin 1:** Input matrix image of type *IMAGE_GREY_F*.

- **Outpin 1:** Output matrix image of type *IMAGE_GREY_F*

Parameters

mode(string)	CYCLIC_QR (default) or QR_BACK_SUB • CYCLIC_QR performs n times the QR factorizatione for an mxn matrix to directly obtain dual basis. This even works for non quadratic matrices with m > n and yields the pseudoinverse in this case. • QR_BACK_SUB only works for quadratic matrices, therefor it is much faster than CYCLIC_QR.

9.14.9 Plugin KMeans

K-Means clustering.

Plugin provides the standard K-Means clustering procedure on the input image interpreted as matrix of data row vector. The first output image is a column of cluster labels, the second output image contains the cluster centers as a matrix of row vectors

- **Inpin 1:** Input image must be of type *IMAGE_GREY_F*. Input is mandatory.

- **Outpin 1:** Contains column of cluster labels, type *IMAGE_GREY_F*
- **Outpin 2:** Contains cluster centers as matrix of row vectors, type *IMAGE_GREY_F*

Parameters

cluster_↩ count(long)	number of clusters. By default it is set to 2.
iteration(long)	number of iterations. By default it is set to 100.
initial_centers	defines how the cluster centers will be initialized. The valid choices are: *RANDOM* and *L↩ AST_COLUMN*, in case of *RANDOM* every data point is assigned to some cluster at random. In case of *LAST_COLUMN* the initial clusters are given be labels in the last column of the input image (the dimension of the input data is then width of input image - 1)

9.14.10 Plugin Multiply

Plugin for matrix multiplication.

This node multiplies two images considered as matrices. The matrix sizes have to be compatible to have a defined matrix product, i.e. the width of the left factor (first input image) has to be the height of the right factor (second input image). If the images are three-dimensional, the matrix product is calculated for each z-plane independently.

- **Inpin 1:** matrix image of type IMAGE_GREY_F
- **Inpin 2:** matrix image of type IMAGE_GREY_F. Height has to be of the same size as width of first input image.

- **Outpin 1:** matrix image of type IMAGE_GREY_F.

9.14.11 Plugin PCA

Principal Component Analysis (PCA).

The plugin performs the Principal Component Analysis (PCA) of the input image. The input image is considered as a matrix where the data is arranged in column vectors.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** First output image of type IMAGE_GREY_F containing the PCA projected data
- **Outpin 2:** Second output image of type IMAGE_GREY_F containing the PCA projection matrix.

Parameters

out_dim(long)	Output dimension. By default it is set to 1.

Keywords:

matrix, PCA, principal component analysis

9.14.12 Plugin QR

Node to perform the QR decomposition.

This plugin performs the QR decomposition on the input image A as matrix.
The first output image contains the resulting matrix Q, the second one the matrix R. Both output images are of type IMAGE_GREY_F.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** First output image of type IMAGE_GREY_F containing the resulting matrix Q.
- **Outpin 2:** Second output image of type IMAGE_GREY_F containing the matrix R.

9.14.13 Plugin RANSAC

Random sample consensus algorithm (RANSAC) on data provided by the input image and model provided by the graph.

Random sample consensus algorithm (RANSAC) on data provided by the input image and model provided by the graph. The input image provides the data (points) to be fitted to the model, every row is a point in n dimensions, the points can be pixel coordinates or more abstract features suitable for the model defined by the graph. The plugin runs the RANSAC algorithm to select part of the data that best fits (is explained by) the given model. At every iteration a random sample from the given data is selected, this is used as the first input of the graph. The second input is the given data. The graph should use the data at first input to estimate parameters of the model, and then evaluate the model for the data given at the second input. A model can be any function of the data points returning scalar values. The values should be small (close to zero) for points satisfying better the model and large otherwise. the best sample is stored and determines the final (best) model. the output image consists of one column, nonzero entries identify data points satisfying the best model estimated.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Inpin 2:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

graph	Specifies the path to the graph that implements the model. The path can hold a classical path to a file or a special string to point to a graph stored in the StaticStorage, e.g. "%ITWM↩ DIR%/bin/utility::StaticStorage/linefit". In this case linefit' is the representative of the stored graph. The graph must take two images as input: The first image contains the sample data points. These data will be set to estimate the model. The second image contains all data points, they will be used to check the validity of the model. Every image row corresponds to a data point The output image must consist of one column in which every entry indicates how well the corresponding data point satisfies the model learned based on the sample data points (first input image). The lower the absolute value the better the match.
iterate(long)	Determines the maximum number of iterations to be applied to find the best solution
rnd_sample_↩ size(long)	Gives the number of samples necessary to compute a valid model, for example to estimate a line we need at least two points, to estimate a plane three or more points are needed etc.

thresh-old(double)	Determines how well the data points must satisfy the model to be accepted as explained by the model.

Literature:

- http://en.wikipedia.org/wiki/RANSAC

See also

RANSAC line fit example.

9.14.14 Plugin RANSACWithModellOut

Node to apply the RANSACWithModellOut algorithm to data provided by the input image and model provided by the graph.

Node to apply the RANSACWithModellOut algorithm to data provided by the input image and model provided by the graph.

The input image provides the data (points) to be fitted to the model, every row is a point in n dimensions, the points can be pixel coordinates or more abstract features suitable for the model defined by the graph.

The node applies the RANSACWithModellOut algorithm to select part of the data that best fits (is explained by) the given model. At every iteration a random sample from the given data is selected, this is used as the first input of the graph. The second input is the given data. The graph should use the data at first input to estimate parameters of the model, and then evaluate the model for the data given at the second input. A model can be any function of the data points returning scalar values. The values should be small (close to zero) for points satisfying better the model and large otherwise. the best sample is stored and determines the final (best) model. the output image consists of one column, nonzero entries identify data points satisfying the best model estimated.

- **Inpin 1:** Data Points to be fitted to the modell, vector of width according to modell needs (M) and length N, i.e. MxN (IMAGE_GREY_F)

- **Outpin 1:** Output vector of length 1XN, with entries 0 and 1, depending if the respective points are inliers(1) of the modell or not(0).
- **Outpin 2:** Winning modell output in the form defined by the subgraph

Parameters

graph	Specifies the path to the graph that implements the model. This graph must take two images as input: The first image contains the sample data points. These data will be used to estimate the model. The second image contains all data points, they will be used to check the validity of the model. The lower the absolute value of the validity check the better the match. The first output has size 1XN, in each column indicating the fit of each Point (input row). The second output contains the respective modell parameters if not modell can be built, the default output modell is an image of length 1x1x1 with entry 1
iterate	Determines the maximum number of iterations to be applied to find the best solution
rnd_sample_size	Gives the number of samples necessary to compute a valid model, for example to estimate a line we need at least two points, to estimate a plane three or more points are needed etc.
threshold	Determines how well the data points must satisfy the model to be accepted as explained by the model.

Literature:

- http://en.wikipedia.org/wiki/RANSAC

9.14.15 Plugin ReplicateColumn

This node is used to replicate a column.

This node replicates a provided column until a desired image size is reached.
There are two input images:
The first input image is the column to replicate. The second input image is the template image to determine the desired size of the output.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, IMAGE_GREY_F
- **Inpin 2:** Input image giving desired size

- **Outpin 1:** Output image of type of first input

9.14.16 Plugin ReplicateRow

This node is used to replicate a row.

This node replicates a provided row until a desired image size is reached.
There are two input images:
The first input image is the row to replicate. The second input image is the template image to determine the desired size of the output image.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, IMAGE_GREY_F
- **Inpin 2:** Input image giving desired size

- **Outpin 1:** Output image of type of first input

9.14.17 Plugin SVD

singular value decomposition

This class offers functions to perform the singular value decomposition (SVD) on the input image A (considered as a matrix).
With SVD, A is decomposed into a product $A = USV'$ where U and V' are orthogonal matrices and S is a diagonal matrix containing the singular values.
There are three output images yielding U, S, and V.

Note

For 3d data the operation is performed for every image plane.

- **Inpin 1:** matrix image A of type IMAGE_GREY_F.

- **Outpin 1:** matrix image U of type IMAGE_GREY_F.
- **Outpin 2:** matrix image S of type IMAGE_GREY_F.
- **Outpin 3:** matrix image V of type IMAGE_GREY_F.

Keywords:

matrix, SVD, singular value decomposition

9.14.18 Plugin ShuffleRows

Shuffle the positions of the rows of two images of equal height, following a uniform distribution.

The row positions are permutated uniformly. If two input images are passed, they are processed independenly using the same permutation. For 3D images, all slices are shuffled separately using the same permutation.

- **Inpin 1:** Input image, optional.
- **Inpin 2:** Another input image, optional. Must have same height.

- **Outpin 1:** Optional output image of same type and size as the first input image if available.
- **Outpin 2:** Optional output image of same type and size as the second input image if available.

Note

Note, that the input images must have the same height.
For 3D images, the rows in the slices are shuffled the same way

Keywords:

shuffle rows, random, lines, change position

9.14.19 Plugin Transpose

Node to transpose an image in matrix interpretation.

This node transposes (i.e. exchanges rows and columns of) an input image interpreted as a matrix. For a given three-dimensional image, each z-slice is transposed as a 2d image.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, or IMAGE_G↩ REY_32.

- **Outpin 1:** Output image of the same type as the input image.

9.15 Morphology Plugins

9.15.1 Plugin BeucherGradient

Beucher gradient a.k.a. morphological gradient.

The Beucher (or morphological) gradient is defined as the difference between dilation and erosion of the input image:

$$\rho_B(f) := (f \oplus B) - (f \ominus B) \ .$$

Here, the structuring element B is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindowHeight+1) ∗ (2∗nHalfWindowDepth+1) such that the central pixel is the pivot element.

- **Inpin 1:** Input image to the gradient (allowed types: IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_G↩ REY_16, IMAGE_GREY_32, or IMAGE_GREY_D).

- **Outpin 1:** The gradient image of the same size and type as the input.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

Keywords:

Beucher gradient, erosion, dilation, mathematical morphology

9.15.2 Plugin BlackTopHat

Class to perform the morphological black top hat on the input image, extracts dark structures.

This function performs the morphological black top hat

$$\mathrm{BTH}(f) := (f \bullet B) - f \quad ,$$

i.e. the difference between a closing of the image f and f. The black top hat can serve for extracting small dark structures. The term "small" is meant with respect to the structuring element. In this function, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindowHeight+1) ∗ (2∗nHalfWindowDepth+1) such that the central pixel is the pivot element.

- **Inpin 1:** Input image to black top hat transform (allowed types: IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, or IMAGE_GREY_D).
- **Outpin 1:** The processed image of the same size and type as the input.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

9.15.3 Plugin Closing

Class to perform the morphological closing filter on the input image.

This function performs the morphological closing,

$$f \bullet B := (f \oplus B) \ominus B \quad ,$$

i.e. a dilation of the image followed by an erosion with the same structuring element. In this function, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindowHeight+1) ∗ (2∗nHalfWindow↩ Depth+1) such that the central pixel is the pivot element.

- **Inpin 1:** Input image to the closure (allowed types: IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GR↩ EY_16, IMAGE_GREY_32, or IMAGE_GREY_D).

- **Outpin 1:** The morphologically closed image of the same size and type as the input.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

9.15.4 Plugin CutHill

Morphological Cut-Hill operator.

Remove regions of higher pixel values compared to the surrounding pixel value.
The cut hill algorithm uses morphological reconstruction by dilation to remove the local maxima from an image. For a binary image it will cut the foreground regions that are not N8-connected to the image border.

- **Inpin 1:** The input image where local maxima will be removed (must be of type IMAGE_GREY_8 or IMAGE_GREY_F).

- **Outpin 1:** The processed image of the same size and type as the input.

Note

Connectivity here is implicitly defined by the implementation of the morphological reconstruction. Since reconstruction will use a square structuring element, that is, the full neighborhood with 8 neighbor aka N8, the connectivity will always go along axis and diagonal directions.

Warning

processes 3d images plane-wise

See also

plugin Reconstruction for more details about morphological reconstruction algorithm
plugin FillHole

Literature:

- Vincent: Morphological greyscale reconstruction in image analysis: applications and efficient algorithms, IEEE Transactions on Image Processing 2(2) 1993, pp. 176-201
  - P. Soille: Morphological Image Analysis, chapter 6: Reconstruction based operators.

Keywords:

morphology, cut hills, connectivity, planewise, slicewise, N8, neighboorhood, planewise, slicewise

9.15.5 Plugin Dilation

Dilation filter aka Maximum filter.

The plugin performs the morphological dilation $f \oplus B$, i.e. the grey value in a pixel is replaced by the maximum over all grey values inside the neighbourhood B of this pixel defined by the structuring element. Here, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindowHeight+1) ∗ (2∗nHalf↩ WindowDepth+1) such that the central pixel is the pivot element.

- **Inpin 1:** Input image, allowed types: IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMA↩ GE_GREY_32, IMAGE_GREY_D, and IMAGE_MONO_BINARY.

- **Outpin 1:** The dilated image of the same size and type as the input.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

Keywords:

dilation, maximum filter, mask, mathematical morphology

9.15.6 Plugin Erosion

Erosion filter aka Minimum filter.

The plugin performs the morphological erosion $f \ominus B$, i.e. the grey value in a pixel is replaced by the minimum over all grey values inside the neighbourhood B of this pixel defined by the structuring element. Here, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindowHeight+1) ∗ (2∗nHalfWindowDepth+1) such that the central pixel is the pivot element.

- **Inpin 1:** Input image, allowed types: IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMA↩ GE_GREY_32, IMAGE_GREY_D, and IMAGE_MONO_BINARY.
- **Outpin 1:** The eroded image of the same size and type as the input.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

See also

Dilation

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

Keywords:

erosion, minimum filter, mask, mathematical morphology

9.15.7 Plugin ExternalGradient

External gradient with a rectangular structuring element where the pivot pixel is in the center.

This function computes the external gradient

$$\rho_B^+(f) := (f \oplus B) - f \ ,$$

i.e. difference between the dilation and the input image f. In this function, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindowHeight+1) ∗ (2∗nHalfWindowDepth+1) such that the central pixel is the pivot element. This operation is somewhat similar to a morphological gradient but because it uses the difference to the original image, only, it yields narrower edges. The external gradient is located outside of the object's edges, therefore resulting regions will be larger than those computed by PInternalGradient.

- **Inpin 1:** Input image for the gradient computation (allowed types: IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, or IMAGE_GREY_D).
- **Outpin 1:** The processed image of the same size and type as the input.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

9.15.8 Plugin FillHole

Morphological Fill-Holes operator.

Removes regions with pixel values which are below the surrounding pixel value.
The fill hole algorithm uses morphological reconstruction by dilation to remove the local minima from an image. For a binary image it will fill the background regions that are not N8-connected to the image border.

- **Inpin 1:** The input image where local minima will be filled (must be of type IMAGE_GREY_8 or IMA↩ GE_GREY_F).

- **Outpin 1:** The processed image of the same size and type as the input.

Note

Connectivity here is implicitly defined by the implementation of the morphological reconstruction. Since reconstruction will use a square structuring element, that is, the full neighborhood with 8 neighbor aka N8, the connectivity will always go along axis and diagonal directions.

Warning

processes 3d images plane-wise

See also

plugin Reconstruction for more details about morphological reconstruction algorithm
plugin CutHill

Literature:

- Vincent: Morphological greyscale reconstruction in image analysis: applications and efficient algorithms, IEEE Transactions on Image Processing 2(2) 1993, pp. 176-201
- P. Soille: Morphological Image Analysis, chapter 6: Reconstruction based operators.

Keywords:
morphology, fill holes, connectivity, N8, neighboorhood, planewise, slicewise

9.15.9 Plugin InternalGradient

Class to calculate the internal gradient of the input image.

This function performs the internal gradient

$$\rho_B^-(f) := f - (f \ominus B) \ ,$$

i.e. difference between the input image f and its erosion. In this function, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindowHeight+1) ∗ (2∗nHalfWindowDepth+1) such that the central pixel is the pivot element. This operation is somewhat similar to a morphological gradient but because it uses the difference to the original image, only, it yields narrower edges. The internal gradient is located inside of the object's edges, therefore resulting regions will be smaller than those computed by PExternalGradient.

- **Inpin 1:** Input image for the gradient computation (allowed types: IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, or IMAGE_GREY_D).

- **Outpin 1:** The processed image of the same size and type as the input.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

9.15.10 Plugin MidrangeFilter

Class to perform the midrange filter with a rectangular structuring element where the pivot pixel is in the center.

This function performs the morphological midrange filter

$$\mathrm{mid}(f) := \frac{1}{2}\left((f \oplus B) + (f \ominus B)\right) \quad ,$$

i.e. the arithmetic mean between dilation and erosion. In this function, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindowHeight+1) ∗ (2∗nHalfWindowDepth+1) such that the central pixel is the pivot element.

- **Inpin 1:** Input to the midrange filter, must be of type IMAGE_GREY_F.
- **Outpin 1:** Filtered image.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

Note

Since an averaging is involved, the input type restriction to IMAGE_GREY_F is required.

9.15.11 Plugin Minimum

Minimum filter on the input image.

Performs the Minimum filter on the input image. The Minimum filter checks which pixel value inside the filter window is the minimum and sets the output pixel values accordingly. The size of the filter window can be set arbitrarily.

- **Inpin 1:** Input image must be of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Outpin 1:** Minimum-filtered image.

Parameters

half_↩ windowsize_x	This parameter determines half of the width of the filter window. It will be casted to long. By default it is set to 2.
half_↩ windowsize_y	This parameter determines half of the height of the filter window. It will be casted to long. By default it is set to 2.

Note

supports 2d images only

9.15.12 Plugin MorphologicalLaplacian

Morphological Laplacian with a rectangular structuring element where the pivot pixel is in the center.

The morphological Laplacian is defined as the difference between the external (ρ_B^+) and internal (ρ_B^-) gradient. This function performs the morphological Laplacian

$$\Delta_B(f) := \rho_B^+(f) - \rho_B^-(f) = (f \oplus B) - 2 \cdot f + (f \ominus B) \ .$$

In this function, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindow↩ Height+1) ∗ (2∗nHalfWindowDepth+1) such that the central pixel is the pivot element. This is a general measure for edge strength.

- **Inpin 1:** Input image of type IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, IMAGE_GR↩ EY_F, or IMAGE_GREY_D.
- **Outpin 1:** Edge-detected image.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003 (only one brief note).
- L. van Vliet, A nonlinear Laplace operator as edge detector in noisy images. Computer Vision, Graphics and Image Processing 45(2), 1999.

Note

Input image must be of type IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, IMAGE_GREY↩ _F, or IMAGE_GREY_D

9.15.13 Plugin MorphologicalShockFilter

Morphological shock filter with a rectangular structuring element where the pivot pixel is in the center.

This function performs the morphological shock filter defined via the equation

$$S_B(f) := \begin{cases} f \oplus B, & \Delta_B f < -\varepsilon \\ f \ominus B, & \Delta_B f > \varepsilon \\ f, & \text{else} \end{cases}$$

That means, in each voxel the function f is left unchanged or replaced by its dilation or erosion depending on the sign of the morphological Laplacian $\Delta_B f$. This way, the shock filter identifies influence zones of maxima and minima. By iterating this procedure, borders between influence zone will appear.

In this function, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindow↩ Height+1) ∗ (2∗nHalfWindowDepth+1) such that the central pixel is the pivot element.

- **Inpin 1:** Image on which to apply the shock filter, must be of type IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, IMAGE_GREY_F, or IMAGE_GREY_D.

- **Outpin 1:** The shock-filtered image.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element
epsilon	small limit for testing floating points to zero

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

9.15.14 Plugin Opening

Class to perform the opening with a rectangular structuring element where the pivot pixel is in the center.

This class performs the morphological opening,

$$f \circ B := (f \ominus B) \oplus B \quad ,$$

i.e. an erosion of the image followed by a dilation with the same structuring element. In this function, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindowHeight+1) ∗ (2∗nHalfWindow↩ Depth+1) such that the central pixel is the pivot element.

- **Inpin 1:** Image to which the opening should be applied, must be of type IMAGE_GREY_8, IMAGE_G↩ REY_16, IMAGE_GREY_32, IMAGE_GREY_F, or IMAGE_GREY_D.

- **Outpin 1:** Morphologically opened image.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

9.15.15 Plugin Reconstruction

Morphological Reconstruction by Dilation resp. by Erosion.

The plugin performs the morphological Reconstruction by Dilation or by Erosion on given marker and mask image. Morphological reconstruction means to iteratively apply geodesic dilation (δ_g) or erosion (ε_g) to an image f restricted to mask image g. Using geodesic dilation or erosions,

$$\delta_g^{(1)} = \delta^{(1)}(f) \wedge g \varepsilon_g^{(1)} = \varepsilon^{(1)}(f) \vee g$$

we iterate one of the two until convergence. Convergence is guaranteed for any image. If reconstruction by dilation is used, the result will always be below the original image ("mask") in every pixel. If reconstruction by erosion is used, the result will always be above the original image ("mask") in every pixel. The effect of these transformations can be very different depending on the choice on the mask image.

- **Inpin 1:** marker image, usually some processed version of the original image (IMAGE_GREY_8 or IM↩ AGE_GREY_F).
- **Inpin 2:** mask image, usually the original image, same size and type as marker image.

- **Outpin 1:** reconstruction image of same size and type as marker image (slicewise processed)

Parameters

method	This parameter determines which reconstruction method will be applied : • DILATION • EROSION By default it is set to DILATION.

Warning

the plugin processes 3d data slice-wise, each image slice will be reconstructed independently

Literature:

- P. Soille, Morphological Image Analysis, Ch. 6, Springer, 1999.

Keywords:

morphology, connectivity, morphological reconstruction, geodesic dilation, geodesic erosion

9.15.16 Plugin SelfDualTopHat

Class to perform the morphological selfdual top hat on the input image.

This function performs the morphological selfdual top hat

$$\mathrm{SDTH}(f) := (f \bullet B) - (f \circ B) \ ,$$

i.e. the difference between a closing and an opening of the input image f. The selfdual top hat can serve for extracting both small dark and bright structures. The term "small" is meant with respect to the structuring element. In this function, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindow↩ Height+1) ∗ (2∗nHalfWindowDepth+1) such that the central pixel is the pivot element.

- **Inpin 1:** Input image in which to detect dark and bright structures, must be of type IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, IMAGE_GREY_F, or IM↩ AGE_GREY_D.
- **Outpin 1:** Resulting image.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.

9.15.17 Plugin Thinning

Class to perform the Thinning filter on the input image.

Performs the Thinning filter on the input image as defined at the following website `http://homepages.↩ inf.ed.ac.uk/rbf/HIPR2/thin.htm`

- **Inpin 1:** Input image must be of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Outpin 1:** Thinning-filtered image.

9.15.18 Plugin WhiteTopHat

Node to perform the morphological white top hat on the input image, extracts bright structures.

This function performs the morphological white top hat

$$\mathrm{WTH}(f) := f - (f \circ B) \ ,$$

i.e. the difference between the input image f and an opening. The white top hat can serve for extracting small bright structures. The term "small" is meant with respect to the structuring element. In this function, the structuring element is a rectangle of size (2∗nHalfWindowWidth+1) ∗ (2∗nHalfWindowHeight+1) ∗ (2∗nHalfWindowDepth+1) such that the central pixel is the pivot element.

- **Inpin 1:** Input image to white top hat transform (allowed types: IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, or IMAGE_GREY_D).
- **Outpin 1:** The processed image of the same size and type as the input.

Parameters

half_↩ windowsize_x	width of the rectangular structuring element
half_↩ windowsize_y	height of the rectangular structuring element
half_↩ windowsize_z	depth of the rectangular structuring element

Literature:

- P. Soille, Morphological Image Analysis, Springer, Berlin, 2003.
- D. Lemire, Streaming maximum-minimum filter using no more than three comparisons per element, Nordic Journal of Computing 13(4), 328-339, 2006.

9.16 Segmentation Plugins

9.16.1 Plugin FastChanVese

Node to calculate a fast segmentation of a 2D image based on an algorithm by ChanVese.

This function calculates a segmentation based on an algorithm by Chan and Vese ("Active Contours Without Edges",2001), but the variational problem is not solved via the Euler-Lagrange Equations. Instead, the energy is calculated directly and it is checked if the energy decreases when a point changes from inside the level set to outside and vice versa. This method was proposed in "A Fast Algorithm For Level Set-Based Optimization", 2002, by Chan and Song and it decreases the computation time dramatically.

The evolving of the contour (the zeroline of ϕ) is calculated by minimizing the following functional:

$$\begin{aligned} F(c_1,c_2,H(\phi)) = \mu \int |\nabla(H(\phi(x,y)))| dxdy + \nu \int H(\phi(x,y)) dxdy \\ + \lambda_1 \int |u_0(x,y) - c_1|^2 H(\phi(x,y)) dxdy \\ + \lambda_2 \int |u_0(x,y) - c_2|^2 H(\phi(x,y)) dxdy \end{aligned}$$

where $H(z)$ is a regularized version of the Heavyside function, $H(z) = \frac{1}{2}(1 + \frac{2}{\pi}\arctan(\frac{z}{\varepsilon}))$, u_0 is the given image and c_1, c_2 are the means of the inner, i.e. outer region.

The two steps of the algorithm:

- Step 1: Initialize - Construct an initial partition of phi, one part for $\phi > 0$, one part for $\phi < 0$
- Step 2: Advance - For each point in the image, if the energy F decreases when we change $\phi(x)$ to $-\phi(x)$, then update this point by $\phi(x) = -\phi(x)$. Otherwise, $\phi(x)$ remains unchanged.

Here the initialization is a circle where the inner region is characterized by $\phi > 0$, and the outer region by $\phi < 0$.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of same type as input image.

Parameters

nu	weight of the area of the inner region
mu	weight of the length of the zero set of the level set function
radius	radius of the circle of the initial phi

x_coord	x-coordinate of the circle of the initial phi
y_coord	y-coordinate of the circle of the initial phi
lambda_1	weight of the approximation of the image in the inner region
lambda_2	weight of the approximation of the image in the outer region
iterations	number of iterations

Literature:

- T. Chan and L. Vese, Active Contours without Edges, IEEE Transactions on image Processing, VOL. 10, NO. 2, 2001
- B. Song and T. Chan, A Fast Algorithm for Level Set Based Optimization, CAM-UCLA, 68, 2002

9.16.2 Plugin FeatureSegmentation

Node to segment an image via split/merge algorithm.

Uses a generic split and merge approach to partition an image into regions of homogenous properties. These properties are parametrized through the use of a given feature plugin, e.g. if a plugin calculating the mean gray value is specified, the split and merge algorithm will partition the image into regions of homogenous gray values, while another plugin calculating texture feature values would result in an image outlining regions of similar textures.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Outpin 1:** Output image of the same type as input image.

Note

The amount of regions to process will grow inversely exponential with the value of *split_threshold* .

Parameters

feature_plugin	Specifies the plugin to use for the underlying segmentation method.
merge_threshold	Parameter of type double. Decides on whether to merge adjacent regions.
split_threshold	Parameter of type double. Decides on whether to split a region into subregions.
minimum_↩ region_size	Specifies the minimum number of pixels for a region to be allowed to be split.

9.16.3 Plugin MultiTh

Node to automatically segment an image into multiple regions. The mutual information between the pixel values of the input image and the pixel labels in the result image is maximized.

Segments the input image into a given number of regions. The regions are defind by thresholding the grey values using a list of thresholds. The thresholds are choosen so as to maximize the mutual information between the original pixel values and the pixel labels in the result image.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, or IMAGE_GREY_16.
- **Outpin 1:** Output image of the same type and size as the input image.

Parameters

method	Determines which measure will be used to quantify the mutual information. It can be chosen out of 1. ENT : Entropy (<H(a) + H(b) - H(a,b)>) 2. SSD : Sum Squared Differences (< (p(a)p(b) - p(a,b))² >)
th_count	Specifies the number of thresholds. *th_count* is equal to the number of resulting image regions - 1.
th_divide_by	All thresholds will be divided by this parameter.

Note

Will be very slow for 16-bit images due to full search through all gray levels

Keywords:

labeling, segmentation, multi-threshold, graylevel, grayvalue, entropy, mutual information, sum of squared differences

9.16.4 Plugin Otsu

Otsu's adaptive global thresholding method.

Otsu's method is designed to separate foreground from background. It selects a threshold so that the variance between the resulting classes is maximized while the within-class variance is minimal. If a second input image is given as a mask image, only the areas marked by it are considered for the computation of the threshold. The parameter *ignore_bg* determines whether the parts not marked by the second image are thresholded or set to *false↩ _value*.

- **Inpin 1:** Input image to be segmented, type *IMAGE_GREY_8* or *IMAGE_GREY_F*.
- **Inpin 2:** Optional mask image to restrict threshold selection to the marked areas; Mask image has to be of the same type as the first input image.

- **Outpin 1:** Binarized output image of the same type as the first input image.
- **Outpin 2:** Threshold used in segmentation, that is, Otsu's threshold multiplied by *factor*.

Parameters

factor	This *factor* will be multiplied to the computed threshold before applying it to the image↩ : If *factor=1*, the threshold is used unchanged. If *factor<1*, the threshold is diminished. If *factor>1*, the threshold is augmented. It will be cast to double. By default *factor* is set to *1*. The *factor* must be strictly positive.
ignore_bg	This boolean value determines in case of a second input image, if the background that is NOT contained in the marker image is ignored from the thresholding and set to *false_value*. By default it is set to true.
true_value	Output pixel value, if image values are larger than the automatically determined threshold. By default it is set to 1.
false_value	Output pixel value, if images value are smaller or equal to the automatically determined threshold. By default it is set to 0.
assume256(bool)	If set, the node will assume that all gray values lie in the interval [0,255]. This is faster. If not set, the algorithm will use the actually present gray values, which is slower. Set this option to false if you know that you may have gray values which do not fill the interval [0,255]. By default it is set to true.
verbose(bool)	prints function feedback to log area. By default it is set to false.

Literature:

- N. Otsu, "A threshold selection method from gray level histograms", IEEE Trans. Systems, Man and Cybernetics, 9, pg. 62-66, 1979.

Keywords:

automatic segmentation, adatpive threshold, global threshold, foreground, background, histogram based, Otsu

9.16.5 Plugin RandomChain

Plugin to simulate a random chain, the result image contains in every pixel the number of times this pixel has been visited during the evolution.

This plugin simulates a simple random chain, a string of connected links. The motion of every link is partially random and partially influenced by the input image. First for every link a random motion step is generated. If this is such that the new position is better (tha value of the input image is lower at this site) it is accepted immediately, if the new position is not better the step is accepted with probability 1-prob_th. Next the links are moved in such a way that the chain stays connected, if they moved too far away from their neighbour they are moved back one pixel. The result image contains in every pixel the number of times this pixel has been visited.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of the same type as the input image.

Parameters

prob_th	- probability threshold, determines with what probability a random move will be accepted, should be between 0 and 1 low values mean that almost all moves are accepted and thus the motion is more random, high values make the motion less random and stronger influenced by the input image. by default it is set to 0.9
iterate	- number of iterations, by default set to 100
hold	chain - if set to true the links are connected and we have a real chain, if set to false the links move freely independent of each other. by default it is set to true.
initial_chain	specifies the initial chain, the possible values are • CONSTANT: all the links are placed at height/2, • RANDOM: the positions of the links are purely random, • DIAGONAL: the links are placed on the image diagonal, • COLUMN_MINIMA: the links are placed at the minima of every column of the input image by default it is set to RANDOM, if the value of initial_chain is none of the above values, CONSTANT is chosen.

Keywords:

random chain, probability

9.16.6 Plugin Threshold

Node to compare two input images pixelwise, if the pixel values of the first one are larger than the values of the second one.

In case of two input images it will be checked pixelwise, if the pixel values of the first input image are larger than the pixel values of the second one and the pixel values of the result will be correspondingly set to an arbitrary value.
In case of only one input image it will be checked pixelwise, if the pixel values are larger than an arbitrary threshold and the pixel values of the result will be correspondingly set to an arbitrary value. The result will be an image consisting only of two values.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional second input image of same size and type as first input image.

- **Outpin 1:** Output image of the same type as the first input image.

Parameters

threshold	Threshold parameter with which the pixel values will be compared, if only one image is given. It will be cast to double. By default it is set to 0.
true_value	Value the pixel values will be set to, if the pixel values of the first image are larger than the pixel values of the second one. It will be cast to double. By default it is set to 1.
false_value	Value the pixel values will be set to, if the pixel values of the first image are not larger than the pixel values of the second one. It will be cast to double. By default it is set to 0.
run_flag(bool)	INTERNAL Indicates whether the node already has run before. By default it is set to false.
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to false.

Keywords:

threshold, binarize, true, false, segmentation

9.16.7 Plugin Watershed

Watershed transform.

This plugin computes the watershed transform. In the resulting label image the detected regions are labeled with value 1, 2, ..., the border pixel between regions are labeled with value 0 (watershed pixels). There is an optional post-processing step to set pixels which have only one neighbouring region to the label of this region.

Note

Plugin works only for 2d images, for 3d images only the first plane is processed

- **Inpin 1:** Input image of type *IMAGE_GREY_8* or *IMAGE_GREY_F*.

- **Outpin 1:** Watershed segmented image of type *IMAGE_GREY_F*. A Pixel value of 0 indicates a watershed, pixel value $>$ 0 labels connected regions.

Parameters

connectivity(string)	Neighborhood configuration (4,8,6.1,6.2).
postprocessing(string)	if "ON" *and connectivity* == 8 watershed pixels with only one neighbouring region are set to the label of this region

Literature:

P. Soille (2003). Morphological Image Analysis: Principles and Applications, 2nd ed.

Keywords:

watershed, segmentation, label image, region, flooding, immersion, grayvalue relief, morphology

9.17 Utility Plugins

9.17.1 Plugin AddMetaData

Node to add/reset single entries to the meta information of the input data.

Add a single entry to the meta ValueMap of the input data by defining the new *key* and its corresponding *value*. If the *key* already exists, the corresponding *value* will be updated.

- **Inpin 1:** Input ValueMap.
- **Outpin 1:** Output ValueMap.

Parameters

key(string)	entry key to be added to/updated in the ValueMap.
value(string)	value to be set at the entry key.

9.17.2 Plugin BoundingBox

Determine bounding boxes in the input image and to draw them into the output image.

Determines bounding boxes in the input image and draws them into the output image. By default the input image will be treated as containing non-zero foreground pixels. In this case, the bounding box for the whole foreground area is being detected. By setting the parameter *label* to *true* the input image will be treated as a label image. In this case, for every labeled region a separate bounding box is created.
The output image can contain either filled or non-filled bounding boxes.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Outpin 1:** Output image of the same type as the input image.

Parameters

bg_↩ value(double)	value for the background, i.e. area not belonging to the bounding box(es), in the output image. By default it is set to 0.
fg_↩ value(double)	value for the pixel belonging to the bounding box(es)) in the output image. By default it is set to 1.
label(bool)	defines whether the input image is treated as a label image or not: • *true:* every labeled region is treated separately and as many bounding boxes as labels are drawn • *false* (default): all non-zero pixels are considered as the image's foreground and one bounding box is drawn.
fill(bool)	defines whether the bounding boxes in the output image are being filled with \fg_value (fill = true) or not (fill = false). By default it is set to *true*.
resize_box(long)	pixel width for resizing the bounding boxes. For *resize_box* $<$ 0 the bounding boxes in the output image will be expanded, otherwise they will shrunk. By default it is set to 0.

Keywords:

draw bounding box

9.17.3 Plugin ConvertType

Change image type and convert pixel data.

Converts the image data to the specified output type by a simple typecast.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, IMAGE_RGB_8, IMAGE_RGB_8I or IMAGE_BINARY_FG.
- **Outpin 1:** Output image of the type specified by the parameter *out_type*.

Parameters

out_type(string)	Type of the output image. Valid entries are IMAGE_GREY_8, IMAGE_GREY_F, IMAG↩ E_BINARY_FG, IMAGE_RGB_8, IMAGE_RGB_8I and IMAGE_RGB_FI. By default it is set to IMAGE_GREY_F.
swap_msb_↩ lsb(bool)	defines whether the endian (byte order) will be changed for non-8-Bit conversion. The endian describes the order of the most significant byte (MSB) and the least significant byte (LSB). By default it is set to *false*.

Warning

if a result looks scrambled when converted from non-8-bit data, make sure that endianess is correctly set, see parameter *swap_msb_lsb*
be beware that integer valued pixel values may overflow, and floating point valued pixel values may truncated/rounded

Keywords:

convert, typecast grayvalues, endianess, little endian, big endian, LE, BE, MSB, LSB

9.17.4 Plugin ConvexHull

Node to determine the n dimensional extrema of a set of points.

It calculates the extreme vertices (vertices of the convex hull) for a set of points in n dimensions, every row specifies one point. The result is a column vector with extreme vertices marked with non zero values.

If two images are given, points given by the first image and lying outside the convex hull of points given by the second image (reference image) are marked.

- **Inpin 1:** Input image of type IMAGE_GREY_F containing point reference coordinates.
- **Inpin 2:** Optional second input image of type IMAGE_GREY_F containing point test coordinates.

- **Outpin 1:** Column vector with extreme vertices marked with non zero values.

Note

Use manipulation::SelectRays together with manipulation::SwitchAxes "ZYX" to extract the corresponding rows from the input image.
Can be used to construct one class classifiers, just compute the extreme vertices of the "good" examples features and use the two image mode to check if a new feature vector lies outside of this so constructed reference convex hull

Parameters

iteration	Optional input parameter. It must be a constant which will implicitly be cast to int. By default it is set to 10. In the STD mode it determines the number of overall iterations and should be set much bigger (more than 10 times) then the number of input vertices. In the SHH mode should be set to small value (like 10) because then it specifies the iterations per vertex.
mode	Specifies the method used to determine the extrema. The default value is set to SHH (separating hyper plane) which is normally faster and better. The alternative STD mode (standard) is kept for compatibility reasons.
min_distance	Determines the accuracy of the SHH method. It should be positive and smaller than one. The default value is set to 0.001. It affects only the SHH mode.

9.17.5 Plugin CopyImage

Plugin copies an input image to an output image.

Copies the input image to the output image. This node supports all image types and sizes.

- **Inpin 1:** Input image of an arbitrary image type.
- **Outpin 1:** Output image containing a copy of the input image.

Keywords:
copy, nop, do nothing

9.17.6 Plugin CurveIndex

Node to compute the index of a curve.

This function computes the index of a curve (integral of angle) for all pixels in the image.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Outpin 1:** Output image of the same type as the input image.

Parameters

method(string)	defines the desired behaviour: • CINDEX: computes the index of the whole curve. • BBOX (default): computes the bounding box of a polynomial approximation of the curve. This method is much faster than CINDEX.

9.17.7 Plugin DrawString

Plugins draws text into the input image.

Puts a text into the input image. Position, size, pixel value, and blending method can all be configured by parameters.

- **Inpin 1:** Input image of type IMAGE_GREY_8 or IMAGE_GREY_F.
- **Inpin 2:** Optional input image of type IMAGE_GREY_F. This image encoding the position coordinates of the texts to be drawn. It must have at least two columns for the *position_x* and *position_y* of a text, if it has a third column then this column will be used as the value of the text. The text given by parameter *text* should have line breaks, in other words, the line number of text should be at least equal to the number of rows of this image. Note that, in the case that this image is given, the parameters *position_x* and *position_y* and possibly the value will be ignored. Also if this image has more than three columns, then the additional columns of this image will be ignored.
- **Outpin 1:** Output image of the same type as the input image.

Parameters

text(string)	text to be written into the image. By default it is set to *IMAGE*. In the case the second input image is given, the text should include line breaks such that the number of non-empty lines of the text is equal or greater than the rows number of second input image.
position_x(long)	x-coordinate of the upper left corner of the text. By default it is set to 0.
position_y(long)	y-coordinate of the upper left corner of the text. By default it is set to 0.
scale(double)	scaling for the size of the text. By default it is set to 1.
value(long)	pixel value to be used for the text. By default it is set to 255.
blending(string)	method to be used for calculating the output pixel value from the input-image and the text string. Available methods are: • OVERLAY (default): text replaces the input pixel grayvalues • XOR: bit-wise xor operation between input pixel grayvalue and text • DARKEN: text value subtracted from the input pixel grayvalue • LIGHTEN: text value added to the input pixel grayvalue

See also

`example for draw many strings`.

Credits:

This plugin uses the FreeType library. For additional information, see `Third Party License` .

Keywords:

draw text, string, font, ASCII

9.17.8 Plugin FindBalancePoint

Node to find the respective balance points of regions in the first input image given as labeled regions in the second input image.

A user-defined number of balance points will be derived from the first input data image
using the labeled regions from the second input image. The balance points are sorted in ascending order by their gray value and the number will be restricted by the number of labels. Based on the parameter *type* the output will either be an image of the same size as the input image with the balance points marked or a list of balance points with their gray values.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Inpin 2:** Label image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

number(long)	number of balance points to be detected. By default it is set to 1.

type(string)	Defines the type of the output image. Possible values are: • STANDARD (default): The output image is filled with zeros and the desired number of balance points are marked with value 255. • SMART: The output image has three rows and *number+1* columns containing the *number* largest balance points sorted in descending order, filled in the following way: – column_0[0] = width of the original image ∗ column_0[1] = height of the original image ∗ column_0[2] = 0 ∗ column_i[0] = x coordinate of the i-th balance point ∗ column_i[1] = y coordinate of the i-th balance point ∗ column_i[2] = gray value of the i-the balance point
verbose(bool)	print function feedback to log area.

See also

FindMax which detects local maxima

9.17.9 Plugin FindMax

Find local maxima in labeled areas of the input image.

A user-defined number of local maxima will be derived from the first input image
using the labeled regions from the second input image as areas of interest. If no label image is provided, the global maximum will be detected. The local maxima are sorted by their gray value in descending order.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Inpin 2:** Optional label image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

Parameters

number(long)	number of local maxima to be detected. By default it is set to 1.
type(string)	Defines the type of the output image. Possible values are: • STANDARD (default): The output image is filled with zeros and the desired number of local maxima are marked with value 255. • SMART: The output image has three rows and *number+1* columns containing the *number* largest local maxima sorted in descending order, filled in the following way: – column_0[0] = width of the original image – column_0[1] = height of the original image – column_0[2] = 0 – column_i[0] = x coordinate of the i-th local maximum – column_i[1] = y coordinate of the i-th local maximum – column_i[2] = gray value of the i-the local maximum
label_↩ type(string)	Type of the label image. Possible values are: • ARBITRARY (default): The labels can be arbitrary floating-point numbers. • INCREASING_NATURAL_NUMBERS: The labels are natural numbers, e.g. for images created with the plugin *Labeling*. In this case the computation is much faster.
verbose(bool)	print function feedback to log area.

See also

FindBalancePoint which detects the balance points of the labeled objects

Keywords:

find maximum, label image

9.17.10 Plugin GetCoordinates

Extract all non-zero pixel coordinates.

The plugin returns the coordinates of masked (that is, non-zero) pixels of the input image w.r.t. to a mask image. If no second input image is given as mask image, the mask is the input itself, that is,the coordinates of all non-zero pixels are extracted. Each row in the output image corresponds to one non-zero pixel and contains its corresponding x- and y-coordinate for 2d, and z-coordinates for 3d. If parameter 'label' has value true, then the grayvalue/labelvalue is prependend as first column.

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Inpin 2:** Optional mask image for selection, type IMAGE_GREY_F, same size as input image.
- **Outpin 1:** Output image of N pixel coordinates, type IMAGE_GREY_F, size 2xN resp. 3xN.

Parameters

label(bool)	defines whether the pixel values (label) from the input image will be stored in the first column of the result image • *true:* result image consists of three columns (label,x,y) • *false:* result image consists of two columns (x,y) by default this parameter is set to false
3d(bool)	use 2d coordinates (false, default) or 3d coordinates

Note

when parameter '3d' is false, only the first slice is processed
if no non-zero pixel is found, the resulting image has a zero size

Keywords:

coordinate table, coordinate vector, extract coordinates, pixel list

9.17.11 Plugin GetImageAttributes

Node to extract the Attributes() ValueMap from an image.

This node extracts a copy of the internal Attributes() ValueMap of the input image.

- **Inpin 1:** Input image of an arbitrary type.

- **Outpin 1:** ValueMap containing a copy of all image attributes.

See also

SetImageAttributes

9.17.12 Plugin Histogram

Node to compute a gray-value histogram of the input image.

A gray-value histogram of the input image with *bin_count* equally distributed bins is created on a user-defined pixel range. The output is a 1d column image of size 1x(bin_count) which gray values correspond to the bin counts.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.

- **Outpin 1:** 1d histogram image of type IMAGE_GREY_F of size 1xN where N is *bin_count*.

Parameters

range_min(long)	lower bound of values to be taken into account. By default it is set to 0.
range_↩ max(long)	upper bound of values to be taken into account. By default it is set to 255.
bin_count(long)	defines the number of histogram bins. By default it is set to 256.
normalize(bool)	defines whether the resulting histogram will be normalized to sum 1. By default it is set to *false*.

Note

Only values >= 0 are taken into account.

See also

Histogram2d for the joint histogram of two images.

Keywords:

histogram, analysis, statistics

9.17.13 Plugin Histogram2d

Computes the joint gray value histogram of two images.

A joint gray value histogram of two input image with *bin_count* equally distributed bins is created on a user-defined pixel range. The output is a 2d image which gray values correspond to the bin counts.

Note

the pixel range and number of bins are assumed to be equal for both images.

- **Inpin 1:** First input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Second input image of the same type as the first input image.

- **Outpin 1:** Output image of type IMAGE_GREY_F containing the 1d joint histogram.

Parameters

range_min(long)	lower bound of values to be taken into account. By default it is set to 0.
range_↩ max(long)	upper bound of values to be taken into account. By default it is set to 255.
bin_count(long)	defines the number of histogram bins. By default it is set to 256.
normalize(bool)	defines whether the resulting histogram will be normalized to sum 1. By default it is set to *false*.

See also

Histogram for the histogram of a single image.

Keywords:

histogram

9.17.14 Plugin Hough

Node to perform the Hough transformation on the input image.

Performs the Hough Transform on the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of the same type as the input image.

Parameters

direction(string)	Specifies in which direction the transformation will be performed. Available options are: • FORWARD (default): Performs the Hough transform. • BACKWARD: Performs the inverse Hough transform.
in_type(string)	specifies the input type For the *BACKWARD* transform: • STANDARD (default): input consists of a regular image • SMART: the input image contains a list of pixels, where each row contains x, y and the pixel value corresponding to local maxima of a Hough transform. Such a list can be created using the Fplugin :FindMax.

See also

HoughCircle to perform the Hough transform on circles.
FindMax to create a list of local maxima.

9.17.15 Plugin HoughCircle

Node to find circles using the Hough transform.

Node to find circles using the Hough transform. If the pixel is potentially a pixel of the circumference (i.e. if the amplitude exceeds a certain *limit*), it draws a cone above this pixel. The intersection of several cones could be the centre of the circle. All pixels in the 3d Houghspace, that exceed a certain *threshold* are given out as potential centres.

After the transform, you may use the node FindMax to find the potential centers with a local maximum in the Houghspace.

- **Inpin 1:** Input image of type IMAGE_GREY_F

- **Outpin 1:** 3d Houghspace image with the cones, type is IMAGE_GREY_F
- **Outpin 2:** 2d image with potential circle centers, type and size are the same as the input image
- **Outpin 3:** 2d image with circle radii which match the respective center point in second output image, type and size are the same as the input image

Parameters

max_↩ radius(long)	maximal radius of the cone and thus decides the size of the 3d Houghspace. By default it is set to 100.
limit(double)	The amplitude must achieve this limit before the cone is drawn - a marginal value. By default it is set to 5.0.
threshold(long)	If a pixel in the 3d Houghspace exceeds this threshold, it is marked as a potential centre in the second output image - a marginal value. By default it is set to 35.
differentiation	(string) Specifies the decision process for a cone to be drawn. It can be one of the following cases: • BINARY (default): the input image is a binary image. The cone will be drawn if the value of the pixel is unequal to 0. • AMPLITUDE: the amplitude of each pixel is computed. If it exceeds the *limit*, the cone will be drawn. • GRADIENT: the gradient of each pixel is computed. If it is higher than the *limit*, the cone will be drawn.

See also

Hough to perform the standard or inverse Hough transform.
FindMax to create a list of local maxima.

9.17.16 Plugin ImageInfo

Node to extract basic information from the input image.

Some basic information about the input image in being exported. The two outputs contain the following information↩
:

- the first output contains a 1x3 image filled with the three dimensions (width, height and depth).
- the secound output is a ValueMap with the same information as the first output plus the image type and (if available) the image path.

- **Inpin 1:** Input image, accepts all image types.

- **Outpin 1:** Output image of type IMAGE_GREY_F containing the image dimensions.
- **Outpin 2:** Output map of type ValueMap with keys 'type', 'width', 'height', 'depth' and (if available) 'image↩
_path'.

Parameters

verbose(bool)	print function feedback to log area..

9.17.17 Plugin ImageToString

Node to write image data into a string.

This plugin writes image data into a string. The data will be separated with blanks. The rows in the image are separated with a newline (*n*).

- **Inpin 1:** Input image of type MAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GRE↩
Y_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** A string containing the data of input image.

See also

SaveASCII to save data to an image file format.
StringToImage to convert data back from string to image.

9.17.18 Plugin Labeling

Plugin labels connected components in the input image.

Each connected component in the input image is uniquely and consecutively being labeled. The considered neighborhood (i.e. the number and configuration of the surrounding pixels considered as connected to the center pixel) can be set to be 4 or 8 by using the parameter *neighborhood*. If image has more than one slice, each slice is processed independently.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8
- **Outpin 1:** Output image of the same type as the input image (or IMAGE_GREY_F if parameter *upcast* is true)

Note

only available for 2d images

Parameters

neighbourhood(long)	neighborhood configuration being considered, valid entries are 4 and 8. By default it is set to 4.
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to false.
verbose(bool)	print function feedback to log area.

Keywords:

labeling, conneced components, segmentation, objects, labels

9.17.19 Plugin MutualInformation

Node to compute the mutual information of a partitioning of the first input image.

The mutual information of a partitioning of the first input image is being computed. The partitioning of the first input image is defined by the second input image.

- Part 1: the pixels in the first input image for which the second input image. is non-zero.
- Part 2: all remaining pixels, i.e. with value zero in the second input image.

The function computes the mutual information of the two parts of the first input image (i.e. a quantity that measures the mutual dependence of the two) and prints the result to the screen. This node does not modify image contents; the input images are copied to the output images.

- **Inpin 1:** First input image, must be of type IMAGE_GREY_F.
- **Inpin 2:** Input image, must be of type IMAGE_GREY_F.

- **Outpin 1:** First output image, a copy of the first input image.
- **Outpin 2:** Second output image, a copy of the second input image.

9.17.20 Plugin Normalize

Normalize pixel values of the input image.

Normalizes the pixel values of the input image. If required, the data can be modified by removing extreme values.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Outpin 1:** Output image of the same type as the input image (or IMAGE_GREY_F if parameter *upcast* is true).

Parameters

out_↩ min(double/long)	minimum gray value in the output image. By default it is set to 0.
out_↩ max(double/long)	maximum gray value in the output image. By default it is set to 255.
out_↩ default(double/long)	output image value in case of equal minimal and maximal input image value. By default it is set to 0.
min_max_↩ map(string)	Minimum and maximum value used for correcting the input data. All input pixel values lower than min are mapped to min, all input pixel values greaterthan max are mapped to max. This parameter is of the form "min max" By default it is set to NULL.
map_↩ min(double/long)	Minimum value used for correcting the input data. All input pixel values lower than *map_min* are mapped to *map_min*.. it is used only if map_min != 0 and map_max != 0. By default it is set to 0.
map_↩ max(double/long)	Maximum value used for correcting the input data. All input pixel values greater than *map↩ _max* are mapped to *map_max*.. It is used only if map_min != 0 and map_max != 0. By default it is set to 0.
tileX(long)	Number of tiles in x-direction, the input image is divided in. The maximum and minimum in every tile is computed and the average of these values is used in the normalization mapping as the maximal and minimal input pixel value. For tileX = 1, tileY = 1 and tileZ = 1 the exact maximal and minimal input pixel values are used. By default it is set to 1.
tileY(long)	Number of tiles in y-direction, the input image is divided in. The maximum and minimum in every tile is computed and the average of these values is used in the normalization mapping as the maximal and minimal input pixel value. For tileX = 1, tileY = 1 and tileZ = 1 the exact maximal and minimal input pixel values are used. By default it is set to 1.
tileZ(long)	Number of tiles in z-direction, the input image is divided in. The maximum and minimum in every tile is computed and the average of these values is used in the normalization mapping as the maximal and minimal input pixel value. For tileX = 1, tileY = 1 and tileZ = 1 the exact maximal and minimal input pixel values are used. By default it is set to 1.
upcast(bool)	Indicates whether the output image is cast to float. By default it is set to false.

Keywords:

normalize, normalise, grayvalue, clipping

9.17.21 Plugin PaintMarks

Plugin paints marks from a given mark file into the input image.

Paints marks from a given mark file into the input image. For all marks in the mark file the correpsonding region of interest is set to one. I.e. the plugin extracts the coordinates from the mark file and generates a mask image (with only 0 and 1 pixels) as described by the specified mark file.

- **Inpin 1:** Input image of type IMAGE_GREY_F or IMAGE_GREY_8.
- **Inpin 2:** Optional string to define the path to the mark file. Alternatively, the parameter *mark_file* can be used to set the path.

- **Outpin 1:** Output image of type IMAGE_GREY_F, with the same size as the input image. With the parameter *downsample* the image size can be modified.

Parameters

mark_file(string)	Optional path to the mark file. This parameter is only being used if no second input is provided. By default it is set to *NULL*.
downsample(long)	downsampling rate for the output image. By default it is set to 1.

Keywords:

marks, paint, draw, mark file, bounding box

9.17.22 Plugin PixelLabel

Node to label every non zero pixel with a unique label.

Each non-zero pixel is being labeled with its order of occurence in the image.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F.

9.17.23 Plugin Radon

Plugin computes the Radon or Hough transform.

Computes the Radon or Hough transform of the input image. The size of the output image is determined automatically. Supports 2d and 3d images.

- **Inpin 1:** Input image of type IMAGE_GREY_F.

- **Outpin 1:** Output image of type IMAGE_GREY_F containing the Radon or Hough space.

Parameters

run_flag(bool)	INTERNAL Indicates whether the node already run before. By default it is set to false.
mode(string)	Determines how the line histogram is being computed. There are two possibilities: • RADON (default) • HOUGH For binary images significantly faster.
raw_out_↩path(string)	Defines the path to the output file the raw points in the line space (distance,angle) will be written to. Only active in the HOUGH mode. By default it is set to NULL.

Literature:

- Radon, Johann (1917), "Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten", Berichte über die Verhandlungen der Königlich-Sächsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physische Klasse [Reports on the proceedings of the Royal Saxonian Academy of Sciences at Leipzig, mathematical and physical section], Leipzig: Teubner (69): 262-277
- Radon, J.; Parks, P.C. (translator) (1986), "On the determination of functions from their integral values along certain manifolds", IEEE Transactions on Medical Imaging, 5 (4): 170-176
- Hough, Paul (1959), "Machine Analysis of Bubble Chamber Pictures", Proc. Int. Conf. High Energy Accelerators and Instrumentation
- Duda, Richard and Hart, Peter (1972), "Use of the Hough Transformation to Detect Lines and Curves in Pictures," Comm. ACM, Vol. 15

Keywords:

Radon transform, Hough transform, sinogram, detect lines, line detector

9.17.24 Plugin RayMaximum

Node to compute the maximum of every ray, that is, every line in the z-direction.

For all (x,y) coordinates in a 3d input image the maximum of the corresponding ray (i.e. line in z-direction) is computed.

- **Inpin 1:** 3d input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_G↩ REY_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** Largest grayvalues projected to 2d image, image has same type, width and height as the input image.

See also

plugin RayMinimum

Keywords:

maximum, ray, z direction

9.17.25 Plugin RayMinimum

Node to compute the minimum of every ray, that is, every line in the z-direction.

For all (x,y) coordinates in a 3d input image the minimum of the corresponding ray (i.e. line in z-direction) is computed.

- **Inpin 1:** 3d input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_G↩ REY_32, or IMAGE_MONO_BINARY.

- **Outpin 1:** Smallest grayvalues projected to 2d image, image has same type, width and height as the input image.

See also

plugin RayMaximum

Keywords:

minimum, ray, z direction

9.17.26 Plugin ReadASCII

Read data from an ASCII-file into 2D or 3D image.

Plugin reads data from an ASCII-file into the output image. The pixel values have to be separated with a non-numerical character. The separator must not be '-' or 'e'. The output image size is determined automatically from the structure of the data in the file:

- Data in one line of the text file is stored in one row of the image.
- With the line separators, the plugin determines the height of the image.
- For 3d data, the planes are separated in z-direction by two empty lines of length 0.
- Lines starting with pound / hashtag symbol '#' are treated as comments and ignored.

- **Outpin 1:** Output image of type IMAGE_GREY_F, 2d or 3d.

Parameters

filename(string)	file name containing the ASCII data. environment variables can be included. By default it is set to *NULL*.
separator(string)	Character used as a separator. Supported characters are: WHITE SPACE, AUTO, COMMA, ., /, ? and ;. The option *AUTO* tries to find the separator automatically. By default it is set to *"WHITE SPACE"*.
run_flag(bool)	INTERNAL Indicates whether the plugin already run before. By default it is set to false.
verbose(bool)	print function feedback to log area.

Note

Environment variables can be used in the filename. The environment variable must be enclosed in percent signs, e.g. "%TMP%/data.ascii". If the environment variable is not set, it will be replaced by an empty string. In the demo version, the image size is restricted to 256x256x8 pixels. If the file contains a larger image, only the cube of size 256x256x8 in the upper left corner is read.

See also

ReadImage to read data from an image file
SaveImage to write data to an image file format.
SaveASCII to write data into an ASCII file

Keywords:

ascii, file, text, load, read, CSV

9.17.27 Plugin ReadImage

Load image from file.

Reads an image from a file and writes it into the output image. Output image size and type are determined automatically from the data to be read.

- **Outpin 1:** Output image, size and type are determined from the given input file.

Parameters

filename(string)	Name of the file containing the image data. Path can include environment variables. By default it is set to *NULL*.

Note

in the filename environment variables can be used. The variable must be enclosed in percent signs, e.g. "%↩TMP%/image.bmp". If the environment variable is not set, the replacement is empty.
In the demo version, the size of the images is restricted to 256x256x8. If the input file contains a larger image, only the cube of size 256x256x8 in the upper left corner is read.

See also

ReadASCII to read data from an ASCII file.
SaveImage to write data to an image file format.
SaveASCII to read data to an image file format.

Keywords:

load, read, import, fileformat

9.17.28 Plugin ReadMap

Plugin for extracting single entries from a given ValueMap.

This plugin copies the ValueMap entry with the given key to the output. If the requested key does not exist in the input map, an exception is thrown.

- **Inpin 1:** Input ValueMap.
- **Outpin 1:** Output value of the selected key.

Parameters

key(string)	The key of the value map entry to be extracted. By default it is set to *"NULL"*.
verbose(bool)	print function feedback to log area.

9.17.29 Plugin ReplaceNonFinite

Node to replace one specific pixel value with another one.

Replaces pixel value which are non finite, which are +infinity, -infinity, and not-a-number (nan).

- **Inpin 1:** Input image of type IMAGE_GREY_F.
- **Outpin 1:** Output image of the same size and type as the input image.

Parameters

pinf_↩value(double)	replacement value for +infinity.
ninf_↩value(double)	replacement value for -infinity.

nan_↩ value(double)	replacement value for not-a-number (nan).

See also

plugin Divide: plugin ReplaceNonFinite is useful after plugin Divide which may set some pixel to non-finite values.

9.17.30 Plugin ReplaceValue

Node to replace specific pixel values with other pixel values.

Replaces one or more pixel values in the input image with other pixel values.

- **Inpin 1:** Input image of type IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, IMAGE_GR↩ EY_F or IMAGE_GREY_D.
- **Inpin 2:** Optional second input image specifying list of pixel values to be replaced. This input image needs to be of the same type as the first input image. First column contains the source values that shall be replaced, the second column contains the destination values replacing the source values. Width needs to be equal to two. If this input is specified, replacement of *src_value* by parameter *dst_value* is ignored.
- **Outpin 1:** Output image of the same size and type as the input image.

Parameters

src_↩ value(double/long)	source value to be replaced. By default it is set to 0.
dst_↩ value(double/long)	destination value replacing the *src_value*. By default it is set to 0.

9.17.31 Plugin RingBuffer

This node creates a ring buffer for images.

Incoming images are appended in a ring buffer to an output image in a given coordinate direction. The size of the ring buffer is fixed. If the buffer is full, the oldest data is removed such that the next incoming image fits in.
This can be used for appending data from a line camera to a 2d image.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16 or IMAGE_G↩ REY_32.
- **Outpin 1:** output image of the same type as the input image.

Parameters

direction(string)	The direction in which incoming image data is appended. Possible values are *X*, *Y*, and *Z*. By default it is set to *X*.
max_size(long)	The maximal size of the output image in the given direction. By default it is set to 256.
verbose(bool)	print function feedback to log area.

9.17.32 Plugin SampleLine

Node to sample 2d image data along a line using Bresenham algorithm.

The plugin samples data from the input image lying on the line segment defined by the points (ax, ay) and (bx, by) using Bresenham algorithm. The pixel distances in the input image are assumed to be 1 in both x- and y-direction. The points on the line are sampled in distance 1.

- **Inpin 1:** Input image to sample from, type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GREY_32, or IMAGE_MONO_BINARY

- **Outpin 1:** 1d output image of type IMAGE_GREY_F. The size in x-direction is determined from the position and orientation of the line in the input image.

Note

This plugin only supports 2d input images, third dimension is ignored.

Parameters

ax	x-coordinate of the first of two points that determine the line. By default it is set to 0.
ay	y-coordinate of the first of two points that determine the line. By default it is set to 0.
bx	x-coordinate of the second of two points that determine the line. By default it is set to 10.
by	y-coordinate of the second of two points that determine the line. By default it is set to 10.
verbose	print function feedback to log area.

9.17.33 Plugin SamplePoints

Node to sample 2d image data at the coordinates given by the second input image.

The plugin samples data from the input image lying at the pair-wise coordinates given by the second input image. Non integer coordinates will be simply cast to long. For points outside the image range the sampled value is set to 0.

- **Inpin 1:** First input image of type IMAGE_GREY_F.
- **Inpin 2:** Second input image of type IMAGE_GREY_F containing the point coordinates.

- **Outpin 1:** 1d output image containing the sampled data points.

Parameters

3d	(boolean): use coordinate triple instead of coordinate pair for extraction. only useful for 3d images

Note

This plugin only supports 2d input images.

9.17.34 Plugin SaveASCII

Write image data into an ASCII-file.

Plugin writes image data into an ASCII-file. The data will be separated by whitespace. The rows in the image are separated by a newline. Planes in 3D image data are separated by two newlines.

- **Inpin 1:** Input image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, IMAGE_GR↩ EY_32, or IMAGE_MONO_BINARY.

Parameters

filename(string)	Name of the output file. Environment variables can be included. If it is set to *STDOUT* the data will be written to the console and can be seen in the message window in ToolIP. By default it is set to *NULL*.
append(bool)	defines whether an already existing output file is overwritten (*false*) or the new data is appended (*true*). By default it is set to *false*.
text(string)	Optional comment to be written into the file. By default it is set to *""*
precision(long)	number of significant digits for floating point values. default is *9* which is full precision, if precision is negative, full precision is taken.

Note

Environment variables can be used in the *filename*. The environment variable must be enclosed in percent signs, e.g. *"%TMP%/data.ascii"*. If the environment variable is not set, it will be replaced by an empty string. This plugin is not available in the demo version.

See also

SaveImage to save data to an image file format.
ReadASCII to read data from an ASCII-file.
ImageToString for converting an image to a string in the same format.

Keywords:

ascii, text, asciifile, textfile, store, save, write, harddisc, export, output

9.17.35 Plugin SaveImage

Node to save an image to an image data file.

The input image data is written to an image data file format. Input format and image format must match.
Example for float images:

- apply the Normalize plugin (Interval: [0,1]) and save as tiff (IO or IO2).
- convert to 8-bit (RGB or grey) and save as png (only IO2!) or respective formats. Note that normalization might be needed before conversion.
- **Inpin 1:** Input image data.

Parameters

filename(string)	Full path to the output file containing the image data, including the image type extensiuon. By default it is set to *NULL*.
IO/IO2(string)	determines which routine is used. See above comments for examples.

Note

in the filename environment variables can be used. The variable must be enclosed in percent signs, e.g. "%↩
TMP%/image.bmp". If the environment variable is not set, the replacement is empty.
This plugin is not available in the demo version.

See also

ReadImage to read image data from an image file.
ReadASCII to read data from an ASCII file.
SaveASCII to write image data to an ASCII-file.

9.17.36 Plugin Script

Run a script on input images.

This plugin executes a given script (via the *system()* call) in the specified path on the data given with the input image and gives the resulting image back to the output image. Note, that input images, output images, and parameters are encoded using a percent sign before the keys in *script_path*.

- **Inpin 1:** Input image.
- **Inpin 2:** Optional second input image.
- **Inpin 3:** Optional third input image.
- **Outpin 1:** Output image.
- **Outpin 2:** Optional second output.

Parameters

script_path	(string) Defining the path and the name of script and parameters of script. It is possibe to type the parameters of script in three input parameters.
temporary_path	(string) Defines the folder for temporary image and files. By default, it is set to empty string. Note that, in this case, the plugin will use default temporary directory in the opreating system.
first_parameter	(string) Determines the first parameter coming after *script_path* .
second_↩ parameter	(string) Determines the second parameter coming after *script_path* .
third_parameter	(string) Determines the third parameter coming after *script_path* .
clean	Determines if the temporary files will be automatically deleted after the execution (true) or they will be left unhandled (false), by default it is set to true (delete all temporary files)

Note

The command called is build from the parameters: script_path,first_parameter,second_parameter and third parameter. The place of the parameters in the command may be indicated with the keys "%p1", "%p2", "%p3".
In script_path and parameters it is possible to use some special keys. The supported keys are:

- "%i1" indicates the first input image from slot 1. "%i2" and "%i3" are for the second input_slot and third input_slot.
- "%i1.sizex", "%i1.sizey", and "%i1.sizez" are the size of image of the first input-slot in x, y and z directions, repectively.
- "%i1.type" is type of image from input-slot 1.
- "%i1.raw" means that the image from the first input-slot will be written in RAW format.
- "%i1.tif" means that the image from the first input-slot will be written in TIFF format (useful for float images).

- "%i1.asc" means that the image from the first input-slot will be written in ASCII format (same as plugin SaveASCII).
  - "%i1.xxx" means that the image from the first input-slot will be written in XXX format e.g., "%i1.jpg" writes the image as JPG file. For the second input_slot and third input_slot all the above keys are valid.

  - "%o1" is for first output-slot and "%o2" is for the second output-slot.
  - "%o1.mark" reads the first output with ReadAscii.
  - "%o1.xxx" tries to read the first output as format xxx

Note

arbitrary user parameters can be used too, they are indicated as "%parameter_name", this symbolic name will be replaced by the actual parameter value which must be of type string, double, long or bool example: if the node has a parameter called *name* with the value *v*, "%name" in *script_path* will be replaced by *v* before the command is executed
In the demo version, the sizes of the images used with plugin Script are restricted to 256x256x8.

Keywords:

script, shell, batch, external, run a program, command line

9.17.37 Plugin SelectImage

Select and copy one of the input images.

This plugin selects either the first or the second input images based on the value of the third input. The third input can either be a single value of type long or bool, or an image of type IMAGE_GREY_8 / IMAGE_GREY_F. If the third input is an image, the value of the first pixel (0,0,0) is considered for the evaluation. In case the value equals zero it is interpreted as false and the second input is selected, otherwise, if the value is non-zero it is interpreted as true and the first image is selected. If the third image is *NULL*, the first image is chosen.

- **Inpin 1:** 'true' input image of an arbitrary type, chosen when selector evaluates non-zero.
- **Inpin 2:** 'false' input image of an arbitrary type, chosen when selector evaluates zero.
- **Inpin 3:** image selector: value of type long, bool or an image of type IMAGE_GREY_8 or IMAGE_GR↩ EY_F.

- **Outpin 1:** Output image of the same type and size as the selected image.

Keywords:

select, image, decision, copy

9.17.38 Plugin SelectPixel

Select pixels from two inputs according to a condition image.

Select pixels from two inputs according to a condition image.

- **Inpin 1:** 'condition' image of an arbitrary type. Pixels are converted to boolean.
- **Inpin 2:** 'then' image of an arbitrary type. Pixel will be selected if 'condition' pixel is true.
- **Inpin 3:** 'else' image of same type as 'then' image. Pixel will be selected if 'condition' pixel is false.

- **Outpin 1:** Output image of the same type as 'then' image.

@keywods select, selection, decision, pixelwise, if-then-else

9.17.39 Plugin SetImageAttributes

Node to extend the attributes of the input image.

Node to extend the attributes of the input image. The attributes are extended and not completely overwritten. The attribute name is determined by the parameter *key* and the attribute value is determined by the parameter *value*. If the attribute already exists it will be overwritten, otherwise it will be added. If key is empty *and* the value is of type map, then the resulting attribute are a copy of the map.

- **Inpin 1:** input image of any type
- **Outpin 1:** input image with the modified attribute[key] = value

Note

Changes only the attributes - does not change the image buffer

Parameters

key(string)	The attribute name to be added/updated. if key is empty *and* value is a map, then the full attribute map is set
value(string)	The attribute's value

See also

GetImageAttributes

9.17.40 Plugin Sleep

Plugin causes a user-specified delay in the graph.

This plugin enables the user to integrate a specific delay at a certain place in the graph. The plugin following the Sleep-plugin starts after the given time to execute.

- **Inpin 1:** No restriction w.r.t. input type.
- **Outpin 1:** Copy of input.

Parameters

time(long)	The time how long the graph execution is delayed at this place.
unit(string)	Unit of *time*. Possible values: seconds, milliseconds.

Keywords:

sleep, delay, time, timeout, seconds, milliseconds

9.17.41 Plugin Sort

Plugins sorts all rows in image by a given column.

Sorts the rows of the input image in ascending order according to the pixel values in the given column. The row with the lowest value will be at the top and the row with the highest value will be at the bottom of the image.

- **Inpin 1:** Input image of type IMAGE_GREY_8, IMAGE_GREY_F, IMAGE_GREY_16, IMAGE_GR↩ EY_32, nor IMAGE_MONO_BINARY.
- **Outpin 1:** Output image of the same type and size as the input image.

Parameters

column(long)	specifies the column to be used to do the sorting. By default it is set to 0.

Note

for 3d images the sorting is performed plane-wise
To keep track of the sorted rows you can first append an index column to the image by using *PositionToValue* and *Append*. This index column can be used to perform the reverse sort on the rows to recover the initial state.

See also

plugins *PositionToValue* and *Append*

Keywords:

sort, row, column, ascending order

9.17.42 Plugin StaticImage

Node to read from or write to an internal static float image.

This plugins allows writing to or reading from an internal static float image. Since the internal image is static it is the same in all instances of the node and can be used as a kind of global variable throughout the application across different graphs.

- **Inpin 1:** Input image of type IMAGE_GREY_F (only being used in the *WRITE* mode).
- **Outpin 1:** Output image of type IMAGE_GREY_F (only being used in the *READ* mode).

Parameters

mode(string)	Defines whether to read or write the static image. the possible values are*WRITE* and *READ:* • WRITE the input image data is written to the internal static image. The old internal image is overwritten with the new data. No output is generated. • READ(default) a copy of the internal static image is written to the output image. If the internal image has not been written to before the result image will have size (0,0,0). No input image is being used.

See also

StaticStorage to read or write ValueMap data from/to an internal map

9.17.43 Plugin StaticStorage

Node to read from or write to an internal static storage.

This node allows for reading from or writing to a global static storage. The storage is the same in all instances of the plugin and can be used as a global variable throughout the application, also across different graphs. In this storage various different datatypes can be stored and referenced by keys. The supported types are:
CImage, CValueMap, CValueVector, CValueBool, CValueDouble, CValueLong, CValueString.

- **Inpin 1:** Input value (only being used in the *WRITE* mode).
- **Outpin 1:** Output value (only being used in the *READ* mode).

Parameters

mode(string)	Defines whether to read or write the static variable. the possible values are: • WRITE the input object is written to the internal static storage and a corresponding key. The old internal data is overwritten with the new data if the key is already in use. • READ(default) a copy of the internal static value corresponding to the key is written to the output. If the internal storage has not been written to before the result storage will only contain the execution time. • DELETE the entry corresponding to the key is removed from the static storage. • CLEAR all entries in the storage. • SHADOW_WRITE any pointer wrapped into CValuePointer can be set to the input that will be saved in the internal static storage, associated by the corresponding key. SHADOW_WRITE create neither a copy of the data referenced by the pointer nor move the ownership of the pointer. Like in WRITE the old internal data is overwritten with the new data if the key is already in use. • SHADOW_READ a copy of the stored pointer corresponding to the key is written to the output. If the internal storage has not been written to before the result ValueMap will only contain the execution time.
key(string)	key for the ValueMap entry to be processed. By default it is set to "".
storage(string)	defines the persistency of the storage: • STATIC: the lifetime of the storage is bound to the plugin. as long as the PStaticStorage plugin/libutitlity.dll is loaded, the storage is present. • REGISTRY: the lifetime of the storage is bound to the basic library/dll. as long as this is loaded, the storage is present, even if the PStaticStorage plugin was unloaded.

See also

StaticImage to read or write image data from/to an internal storage

9.17.44 Plugin Statistics

Global grayvalue statistics.

Computes the average and standard deviation of the input image.

- **Inpin 1:** Input image of type IMAGE_GREY_8 or IMAGE_GREY_F.
- **Outpin 1:** Average gray value (double).
- **Outpin 2:** Standard deviation of the gray values (double).

Parameters

verbose(bool)	print function feedback to log area.

Keywords:

grayvalue statistics, features, average, mean, variance, standard deviation, sdev

9.17.45 Plugin StringManipulation

String manipulation tools.

Plugin reads the optional input string and after evaluation the expression parameter writes it in the output port as a string. The input string is determined with the reserved string i1 surrounded by the percent sign `%`, i.e., "%i1%" in the expression parameter. Also, it is possible to define new parameter. In this case, it is possible to use this parameter name surrounded by `%` in the expression parameter. For example:

- with input string "/Folder/of/images/lena.pgm" and separator "/", then
  - the expression "%i1[4]%" will result in the output string "lena.pgm",
  - the expression "%i1[-1]%" will result in the output string "lena.pgm",
  - the expression "%i1[1:2]%" will result in the output string "/of/images",
  - the expression "%i1[1:2]%/OutImg.jpg" will result in the output string "/of/images/OutImg.jpg".
- with input string "/Folder/of/images/lena.pgm", separator "", and new parameters with the name "key1" and value ".jpg", then the expression "%i1[0:10]%/result/folder/%i1[12:15]%_output%key1%" will result in the output string "/Folder/of/images/result/folder/lena_output.jpg".

- **Inpin 1:** Optional input string to be manipulated. In the case of no input string the output will be the expression parameter itself.

- **Outpin 1:** Output string based on the input string and the expression.

Parameters

expression(string)	expression for the output string. By default it is set to "%i1%".
skip_empty_parts(bool)	If it is set to false then by splitting the string, the empty entries contribute in the result string. By default, it is set to false.
verbose(bool)	Determines, whether some information is printed out on the screen.
separator(string)	Delimiter for splitting the input string. By default, it is the empty string "". In this case the input string will be splitted character by character.

See also

`String Manipulation Example Graph`.

Keywords:

string, substring, expression, splitting, character, delimiter

9.17.46 Plugin StringToImage

Node to read numerical data from a string into the output image.

Reads numerical data from a string and writes it into the output image. The data has to be separated with blanks. The output image size is determined automatically from the structure of the data in the input string, where a newline ($\backslash n$) creates a new row in the image.

- **Outpin 1:** Output image of type IMAGE_GREY_F containing the data from the string.

Parameters

data_↩ string(string)	Containing the image data. By default it is set to 1 0 0 \n 0 1 0 \n 0 0 1

See also

ImageToString for converting an image to a string in the same format.

9.17.47 Plugin TextSerializer

Plugin serializes a given value map to a string.

This plugin serialises a value map and writes it to a string.

- **Inpin 1:** input ValueMap

Parameters

append	This parameter contains an optional string to append to the output.
prepend	This parameter contains an optional string to write in front of the output.
linefeed	If this boolean is false, a linefeed is appended at the end, otherwise not.

- **Outpin 1:** output ValueMap

9.17.48 Plugin TimerStart

This plugin stores the current time as CValueVector.

To use this plugin just connect a PTimerStart with a PTimerStop. The former one stores an ITWM::CTimer object as CValueVector on the stack which can be read from the PTimerStop-plugins.

To enable synchronizing with other plugins, this plugin uses the following stack entries:

- **Inpin 1:** original image

- **Outpin 1:** time of calculation start, generated by this plugin
- **Outpin 2:** a copy of the input image

9.17.49 Plugin TimerStop

This plugin reads the time stored on the stack by the PTimerStart-plugin and calculates the difference.

To use this plugin just connect a PTimerStart with a PTimerStop. The former one stores an ITWM::CTimer object as CValueVector on the stack which can be read from the PTimerStop-plugin.

To enable synchronising with other plugins, this plugins uses the following stack entries:

- **Inpin 1:** start time of calculation, generated by the CTimerStart-plugin, pinout{1}
- **Inpin 2:** input image

- **Outpin 1:** a copy of the input image, received via pinin{2}

9.17.50 Plugin ZoneList

Plugin computes bounding box information (aka zone list) from a labeled image.

Plugin creates a zone list (i.e. a list of regions of interest or bounding boxes) from a labeled image.

The zone list can be stored as a value vector with following entries for each label:

- label: the label number of the region
- ax,ay: the upper left coordinates of the region
- bx,by: the lower right coordinates of the region
- width, height: the dimension of the region
  The second possibility is to store the zone list in an image of type IMAGE_GREY_F with size 4xNumber↩ OfZonesx1. Each row represents a zone, and the coordinates are arranged as (ax, bx, ay, by). Such an image can serve as input of the Box-plugin to draw the zones into an image.
- **Inpin 1:** Label image of type IMAGE_GREY_F, IMAGE_GREY_8, IMAGE_GREY_16, or IMAGE_G↩ REY_32.
- **Outpin 1:** Zonelist as image table or value vector.

Parameters

output_↩ mode(string)	Determines the output mode or the for the zone list. Possible values are VALUEVECTOR and IMAGE. Both modes are described above. Mode IMAGE has the advantage that it can be used as second input of a Box plugin to draw the regions of interest. By default it is set to VALUEVECTOR.
x_offset(long)	Augments the size of a zone to the left and right by x_offset. By default it is set to 0.
y_offset(long)	Augments the size of a zone to the top and bottom by y_offset. By default it is set to 0.

Note

for IMAGE_GREY_F input, pixel values are casted to long, decimal remainder is not taken into account IMAGE_GREY_F input, background pixels are pixels with value long(floatpixelvalue)==0

Keywords:

bonding box, label, coordinates

9.18 Sysutils Plugins

9.18.1 Plugin Console

Print to console.

Writing the input value to the console. Input may be a string, a numeric value, or a map or vector.

- **Inpin 1:** value to print

Parameters

linefeed(bool)	If linefeed=true, a new line is started after plugin's output. Default: false.

Keywords:

console, print, debug

9.18.2 Plugin EnvVar

Accessing Environment variables.

Accessing Environment variables.

- **Outpin 1:** string value of the accessed environment variable
- **Outpin 2:** boolean success state of operation

Parameters

mode(string)	SET/GET/UNSET the environment variable
variable(string)	name of the environment variable
value(string)	in case of mode SET, the value to bet set

Keywords:

environment variable, envvar

9.18.3 Plugin FileExists

Check if a file exist.

Plugin for checking a file exist (and is accessible).

- **Inpin 1:** filename(string) path to file

- **Outpin 1:** existence(bool) TRUE if file exists, FALSE otherwise

Keywords:

fileexists, path, existence

9.18.4 Plugin Platform

Platform information.

Provides platform information.

- **Outpin 1:** operating system: "Windows", "Linux", or "Unkown" (string)
- **Outpin 2:** 32bit or 64bit platform? return bytes: 32 or 64 (long)

Keywords:

platform, operating system, OS, Windows, Linux, 32bit, 64bit

9.18.5 Plugin ReadTextFile

Reads the text from the given text file.

Reads the text from the given text file.

- **Outpin 1:** Text of the text file as vector or string. 'i-1' the entry of the vector is 'i' the line of the text file.

Parameters

start_line_↩ number(long)	Line number to begin reading. if 0, it reads from the first line. default is 0.
number_of_↩ lines(long	Total number of lines to read starting from *start_line_number*. if 0, it reads up to end of the file. default is 0.
filename(string)	Path of the text file.
linewise(boolean)	Read linewise (result will be a vector) or full text (result will be string containing line breaks). default is false.

9.18.6 Plugin SaveText

Write value objects into a text file.

Write input value object (bool/long/float/string/vector/map) to a text file in a serialized form. This means that any contents of the value object will be written one after the other.

- **Inpin 1:** The value to be serialized into a text file

Parameters

filename(string)	The name of the output file.
append(bool)	If true, will append to an existing file, overwrite otherwise. By default, it is set to false.
verbose(bool)	If true, it prints the value also into the console. By default, it is set to false.

Keywords:

IO, I/O, write, save, export, file, value, text, ASCII

9.19 Wavelets Plugins

9.19.1 Plugin ModulusMaxima

Multiscale modulus maxima of wavelet transforms.

Performs wavelet transforms modulus maxima on the inputs image. This method is useful for dicrimination of different types of edges. The theory of this method can be found in the article S. Mallat et al. This algorithm is very similar to Canny edge detection but uses wavelets instead of partial derivatives.

Note

Inputs image must be of type IMAGE_GREY_F.
The first and second input images are the wavelet transforms in X and Y directions, respectively.

- **Inpin 1:** Wavelet transform in x-direction (WTX).
- **Inpin 2:** Wavelet transform in y-direction (WTY).

- **Outpin 1:** Edge strength measure after non-maxima suppression.
- **Outpin 2:** Closed edge contours after applying hysteresis thresholding.

Parameters

m_Angle←↩ Tolerance	is the tolerance value of the angle of wavelet transforms (WTX, WTY) in the plane (X,Y). It is given in in degrees. By default it is set to 15.
m_threshold_←↩ low	is the lower bound for hysteresis thresholding.
m_threshold_←↩ high	is the upper bound for hysteresis thresholding.

Literature:

- S. Mallat et al. "Characterization of signals from multiscale edges", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, No.7, pages 710-732 in July 1992.

9.19.2 Plugin MultiscaleWavelet

Class to performs a multiscale edge detection based on wavelet transforms.

Performs multiscale edge detection based wavelet transforms on the input image. This method is useful for dicrimination of different types of edges. The theory of this method can be found in the article "Characterization of signals from multiscale edges" by S. Mallat et al..

- **Inpin 1:** Square input image of type IMAGE_GREY_F, side length must be a power of 2.

- **Outpin 1:** Smoothed image.
- **Outpin 2:** Wavelet transforms in X direction (edges).
- **Outpin 3:** Wavelet transforms in Y direction (edges).
- **Outpin 4:** Modulus (norm of transforms in X and Y directions), edge strength.
- **Outpin 5:** Edge direction in polar coordinates (radian) in every pixel.

Parameters

nScaleMax	is the last scale. It maust be greater than 0 and less or equal than log_2(N), where N is minimum of number of pixels in x and y directions.

Literature:

- S. Mallat et al. which appeared in IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 14, No.7, pages 710-732 in July 1992.

9.19.3 Plugin OrthoFWT

Class to perform the fast wavelet transform with orthogonal wavelets.

Performs the fast discrete wavelet transform with orthogonal wavelets [Mal98] for the input image. The transform considers the periodic extension of the image in all axes' directions and uses tensor product wavelets in higher dimensions. The ordering of the wavelet coefficients in the output image is given by the so-called nonstandard decomposition [GW02]. That means, the wavelet coefficients are stored in 2D as

d	c_h
c_v	c_d

where d are the low-pass coefficients, and c_h, c_v, c_d are the high-pass coefficients in horizontal, vertical and diagonal direction, respectively. In 3D, this is extended analogously. The output image has the same number of dimensions as the input image while the size can increase: The output is the smallest vector, square, or cube where the side-length is a power of two such that the input image fits in it. The coefficients corresponding to the coarse structures can be found starting at the (0,0) coordinate in the resulting image.

List of supported wavelet types:

- Haar: Piecewise constant function with small support, two coefficients.
- Daub4: Daubechies filters with n=4 coefficients and p=2 vanishing moments.
- Daub6: Daubechies filters with n=6 coefficients and p=3 vanishing moments.
- Daub8: Daubechies filters with n=8 coefficients and p=4 vanishing moments.
- Daub10: Daubechies filters with n=10 coefficients and p=5 vanishing moments.
- Daub12: Daubechies filters with n=12 coefficients and p=6 vanishing moments.
- Daub14: Daubechies filters with n=14 coefficients and p=7 vanishing moments.
- Daub16: Daubechies filters with n=16 coefficients and p=8 vanishing moments.
- Daub18: Daubechies filters with n=18 coefficients and p=9 vanishing moments.
- Daub20: Daubechies filters with n=20 coefficients and p=10 vanishing moments.
- Coiflet1: Coiflet with N=6 coefficients.
- Coiflet2: Coiflet with N=12 coefficients.
- Coiflet3: Coiflet with N=18 coefficients.
- Coiflet4: Coiflet with N=24 coefficients.
- Coiflet5: Coiflet with N=30 coefficients.
- Symmlet4
- Symmlet5
- Symmlet6
- Symmlet7
- Symmlet8
- Symmlet9
- Symmlet10

- **Inpin 1:** Input image must be of type IMAGE_GREY_F.
- **Outpin 1:** Transformation result, coarse structures start at (0,0), coefficients are arranged in nonstandard decomposition [GW02].

Parameters

wavelet_type	This parameter determines which orthogonal wavelet should be used. The default value is Haar. The wavelet coefficients are taken from [Mal98]. See the list of supported values.
coarsest_level	Number of the coarsest level to consider. The wavelet transform always starts with the finest possible level (this has the number 1) and proceeds to the coarsest given level. The default value is -1, i.e. the wavelet transform uses all possible levels.

Literature:

- [Mal98] S. Mallat, A Wavelet Tour of Signal Processing, 2nd edition, Academic Press, San Diego, 1998.
- [GW02] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 2nd edition, Prentice Hall, 2002.

See also

ITWM::CPOrthoIWT ITWM::Wavelets::IWT_PO

9.19.4 Plugin OrthoIWT

Class to perform the fast wavelet transform with orthogonal wavelets.

Performs the inverse discrete wavelet transform with orthogonal wavelets for the input image [Mal98]. It is assumed that the input image has side-lengths which are powers of 2 and is arranged in non-standard decomposition [GW02].

List of supported wavelet types:

- Haar: Piecewise constant function with small support, two coefficients.
- Daub4: Daubechies filters with n=4 coefficients and p=2 vanishing moments.
- Daub6: Daubechies filters with n=6 coefficients and p=3 vanishing moments.
- Daub8: Daubechies filters with n=8 coefficients and p=4 vanishing moments.
- Daub10: Daubechies filters with n=10 coefficients and p=5 vanishing moments.
- Daub12: Daubechies filters with n=12 coefficients and p=6 vanishing moments.
- Daub14: Daubechies filters with n=14 coefficients and p=7 vanishing moments.
- Daub16: Daubechies filters with n=16 coefficients and p=8 vanishing moments.
- Daub18: Daubechies filters with n=18 coefficients and p=9 vanishing moments.
- Daub20: Daubechies filters with n=20 coefficients and p=10 vanishing moments.
- Coiflet1: Coiflet with N=6 coefficients.
- Coiflet2: Coiflet with N=12 coefficients.
- Coiflet3: Coiflet with N=18 coefficients.
- Coiflet4: Coiflet with N=24 coefficients.
- Coiflet5: Coiflet with N=30 coefficients.
- Symmlet4
- Symmlet5
- Symmlet6
- Symmlet7
- Symmlet8
- Symmlet9
- Symmlet10

- **Inpin 1:** Wavelet coefficients in non-standard decomposition [GW02], image must be square with side lengths being a power of 2. Input image must be of type IMAGE_GREY_F.

- **Outpin 1:** Reconstructed image.

Parameters

wavelet_type	This parameter determines which orthogonal wavelet should be used. The default value is Haar.
coarsest_level	The level of the given wavelet transform. Per default, this value is -1, and the algorithm takes the logarithm of the side length of the input file as coarsest level.

Literature:

- [Mal98] S. Mallat, A Wavelet Tour of Signal Processing, 2nd edition, Academic Press, San Diego, 1998.
- [GW02] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 2nd edition, Prentice Hall, 2002.

See also

ITWM::CPOrthoFWT ITWM::Wavelets::IWT_PO

9.20 Display Plugins

9.20.1 Plugin Display

Displays an image.

Plugin to display images and values. For the image display a variety of parameters can be set in the parameter window. Parameters for the image display, the loop functionality, the display of status line features and a number of settings for the algorithm widget are available. This widget can be used to pre-view the influence of a specific algorithm to the current image. AlgorithmWidget(map) allows to apply any loaded *TLP-graph* to an area of a loaded image on the fly. The area has the size of AlgorithmWidget/Width x AlgorithmWidget/Height pixels and has to be chosen by key combination Ctrl+Shift+clicking on the loaded image. This widget also can be used for normalizing of certain image areas by setting the parameter AlgorithmWidget/Normalize. By using the Shift+clicking you will see just the zoomed area.

- **Inpin 1:** image or value

Parameters

Image/↩ Zoom(string)	Describes the zoom factor for the image display, e.g. 100%, see note for more information.
Image/↩ Plane(long)	Choose which plane of a 3d image you want to look at. You can also change the plane in the display window.
Image/↩ Centered(bool)	Determines whether the image should be centered in the display window. By default this is set to *false*, which leads to the image being positioned top left.
Image/↩ Normalize(bool)	This will normalize the image for the visualisation.
Image/ShowAs↩ Table(bool)	Determines whether the image is shown as a table, where the table rows correspond to the image rows and the table columns correspond to the image columns. Each table entry is the value of the image pixel in its position. By default this is set to *false*. You can also switch between the two display formats in the display window by clicking on the button "Show as table".
Image/Table↩ Precision(long)	Determines how many decimal places are shown in the table format. By default it is set to 2.

Status↩ Line(string)	Shows the current information below the displayed image. Supported status line entries are:
	w = width
	h = height
	d = depth
	s = size (wxh for 2-dimensional and wxhxd for 3-dimensional images)
	t = image types are IMAGE_GREY_8, IMAGE_GREY_F, IMAGE_RGB_8, etc.
	imin = minimal image value
	imax = maximal image value
	imean = mean image value
	isigma = standard deviation of the image
	pmin = minimal plane value
	pmax = maximal plane value
	pmean = mean plane value
	psigma = standard deviation for plane
	| = starts new status bar box.

ZoomWidget/↩ Width(long)	Width for the loop function.
ZoomWidget/↩ Height(long)	Height for the loop function.
ZoomWidget/↩ Zoom↩ Factor(double)	Zoom factor for the loop function.
Algorithm↩ Widget/Path	Loads a .tlp-graph.
Algorithm↩ Widget/Zoom	Zooms the area on which you are clicking before applying the chosen algortihm.
Algorithm↩ Widget/Width	Width of the area to which the algorithm should be applied.
Algorithm↩ Widget/Height	Height of the area to which the algorithm should be applied.
Algorithm↩ Widget/↩ Normalize	This will normalize the area of the image to which the algorithm widget is applied.
Thumbnail↩ Widget/Visible	If set to *true*, a thumbnail view window opens additionally to the display window. In this window, a red box on the full image shows which image section you are currently seeing in the display window. By default this is set to *false*. You can also open and close the thumbnail view in the display window by clicking on the button "Thumbnail Preview".
Thumbnail↩ Widget/Max↩ Width	
Thumbnail↩ Widget/Max↩ Height	

Note

The Image/Zoom parameters are admitted in a certain way. In the custom tab you can see which values are allowed for the zoom.

Keywords:

display, show image, sliceview

9.20.2 Plugin Load

Node opens a file dialog and loads a CImage at its output stack.

The node Load can be used to load images. By starting the plugin the Open Image dialog window opens where a image file can be loaded. Currently supported image types are ∗.ppm, ∗.pgm, ∗.jpg, ∗.bmp, ∗.tif and ∗.iass. By default the shown files are restricted to the supported image types.

Parameters

path	Shows the path of the currently loaded image.

9.20.3 Plugin View

Display an image.

Plugin to view an image. For the image view, the parameters described below can be set in the parameter window or directly in the view image window.

Parameters

Normalize	If this box is activated the image will be normlised for visualisation
Zoom_widget	
Width	The width for the loop function
Height	The width for the loop function
Zoom	Describes the zoom factor for the image display

Keywords:

display, view, show

9.21 Plot Plugins

9.21.1 Plugin Plotview

Plot display.

Display the data as plots. Depending on the setting of *data_mode*, either rows or columns will be plotted as a curve. If several rows or columns exist, sevaral curves will be plotted.

- **Inpin 1:** Data to be plotted in form of a row or column-table inside an image. The input image should be of type IMAGE_GREY_F, IMAGE_GREY_8, or IMAGE_MONO_BINARY.

Parameters

pen_size(long)	Line thickness for drawing, default is 1.
curve_↩ style(string)	one of the following curve styles: "Lines" (default), "Sticks", "Steps", "Dots".
curve_↩ title(string)	Title of curve. If there are more than one curve then the name of curves can be given by using a semicolon separated string.
enable_↩ zoom(bool)	Enable zooming, default is true.
zoom_↩ factor(double)	Default is 1.5.
window_↩ title(string)	Title of window, default empty.
x_axis_↩ title(string)	Label for horizontal axis, default is "x".
y_axis_↩ title(string)	Label for vertical axis, default is "y".

x_min_↩ is(double)	value for first x pixel for proper x-axis value assigning (min world coordinate of x), see *x_↩ max_is*
x_max_↩ is(double)	value for last x pixel for proper x-axis value assigning (max world coordinate of x), if *x_↩ min_is* == *x_max_is*, then the pixel position is used
enable_↩ panning(bool)	Enable panning (shifting of plotting area), default is true.
minimum_↩ window_size_↩ x(long)	minimal horizontal plot area size.
minimum_↩ window_size_↩ y(long)	minimal vertical plot area size.
no_of_↩ curves(long)	maximal number of plotted curves. By default, it plots maximum 10 curves. To show all curves, set to zero.
data_↩ mode(string)	can be chosen from *AUTO*, *ROW_BY_ROW* or *COLUMN_BY_COLUMN* • *AUTO:* if the number of rows of input data is more than number of its clumns then input data would be plotet row by row. Otherwise, the input data would be ploted column by column. • *ROW_BY_ROW:* data would be plotet row by row. • @ a COLUMN_BY_COLUMN: data would be plotet column by column.

Credits:

This plugin uses the QWT library. For additional information, see Third Party License .

9.22 Toolima Plugins

9.22.1 Plugin SelectionTool

A simple GUI which allows manual, real time labeling.

This node offers a simple GUI which allows manual, real time labeling in Toolip but also as a standalone application.

– BEGIN ONLY FOR PLUGIN –

- **Inpin 1:** The image which should be annotated.
- **Inpin 2:** An existing image mask.
- **Inpin 3:** A list with all labels.
- **Inpin 4:** A zone list characterizing image masks

Parameters

keep_↩ workspace(bool)	determines whether the workspace should be forgotten after closing the window (set to *false*) or kept (set to *true*)
input_image_↩ path	(string) determines the image path, so that you can save your changes within ToolImA as a ToolImA-project • **Outpin 1:** An image mask containing the ID of the label for each pixel. • **Outpin 2:** A list with all labels. – END ONLY FOR PLUGIN – Selectionbar below the images: The images are organized as a list. You can browse through them by clicking the buttons -> for previous and <- for next. The image's name is depicted here too. If you click on the name, you can choose a different one. Next to the name, 'i of n' says that the shown image is the i-th image of n images in the project. You can add and delete images by clicking on + and x, resp. This changes the number n of images in the project.

To the left, you find buttons for customizing a filter and updating the filter. Next to this, 'i of n' says that the shown image is the i-th image of n images in the filter. You can show more images and omit them by changing the filter settings.

Settings:

Camera memory
The Camera memory remembers the zoom
Don't remember: Every time you return to the image, it will be zoomed such that it fits the window.
Remember each image: The first time you come across it, it is zoomed such that it fits the window. If you zoom in or out, it will be remembered the next time you come across it, as long as you do not close the window.
Remember each project: All images of the project will have the same zoom.

Status Line
Here, you can customize your status line, which shows the current information below the displayed image. Supported status line entries are:
w = width
h = height
d = depth
s = size (wxh for 2-dimensional and wxhxd for 3-dimensional images)
t = image types are IMAGE_GREY_8, IMAGE_GREY_F, IMAGE_RGB_8, etc.
imin = minimal image value
imax = maximal image value
imean = mean image value
isigma = standard deviation of the image
pmin = minimal plane value
pmax = maximal plane value
pmean = mean plane value
psigma = standard deviation for plane
| = starts new status bar box.
As a default, it is set to "(min=%imin max=%imax mean=%imean sd=%isigma)| %t | (%s)" The left-hand side is fixed to [{mouse position in x}, {mouse position in y}, {mouse position in z}] = {pixel value}

Reannotation: You can reannotate your labels within the reannotation window by creating reannotation rules, which match a shortcut key with a certain label. By default, the delete-key is a shortcut for deleting items. You can set the keys within a dialog which opens upon clicking the button on the left below the image display (within the reannotation window). Here, you can enter key combinations manually or record them by clicking the green "+" next to the entry line. This ensures the correct format of the key combinations. After clicking on the green "+", it becomes a red "x". When you click on this, you stop recording and can enter key combinations manually again. When clicking on the bin on the right, you delete the shortcut. The set keys are displayed at the bottom of the reannotation window.

Chapter 10

Summary

In this manual, we introduced the software **ToolIP** and showed many ways of working with its integrated image processing library. This first steps with ToolIP show its potential and its flexibility - and thus help you designing solutions for image analysis problems.
In case of any questions concerning or problems with **ToolIP** and this manual, please do not hesitate to contact us:

`toolip@itwm.fraunhofer.de`

We appreciate to receive feedback and suggestions!

Appendix A

Shortcuts

In the following table the keyboard and mouse shortcuts of **ToolIP** are summarized:

Shortcuts	Description
`Ctrl + N`	open a new workspace
`Ctrl + O`	open graph file
`Ctrl + S`	save graph
`Ctrl + Shift + S`	save graph as
`Ctrl + W`	close current graph
`Ctrl + Z`	undo last action
`Ctrl + Y`	redo last undone action
`Ctrl + A`	select all nodes in the current graph
`Ctrl + C`	copy selected items into clipboard
`Ctrl + X`	cut selected items (remove items and copy them into clipboard)
`Ctrl + V`	paste items from clipboard into current graph
`Ctrl + D`	duplicate selected items into current graph (copy&paste)
`Ctrl + left-click`	on item: add item to selection
`Ctrl + left-click`	on output pin: open visulation on data (see Settings, first visualisation plugin)
`Ctrl + right-click`	on output pin: open visulation on data (see Settings, second visualisation plugin)
`left click + mouse moving`	rectangular selection
`Delete`	delete selected items from the current graph
`Insert`	open fast insertion box to look for a node (updates selection while typing, navigation by Up and Down key, add selected plugin into current graph via drag&drop or by `Enter`)
`Ctrl + F`	open plugin search window for finding a node by attributes in the workspace
`Ctrl + T`	activate/deactivate time measurement
`F2`	rename selected node and set tooltip
`right click`	open context menu
`left double click`	on item in workspace: rename item and set tooltip

Appendix B

Credits for Third Party Libraries

ToolIP uses a number of third party libraries, each released under its individual license terms, see below. The copyright of these libraries remains with their original authors, see below. Note, however, that this does not affect the license terms of **ToolIP** itself in any way. If you would like to obtain the source code of any of the open source libraries used within this software, please contact Fraunhofer ITWM either by mail or e-mail. Please state specifically for which one of the open source third party libraries you require the source code and where it should be sent to.

Fraunhofer ITWM
Markus Rauhut
Fraunhofer Platz 1
67663 Kaiserslautern
toolip@itwm.fraunhofer.de

The following open source third party libraries are used, and their respective authors are acknowledged for their work:

- Boost under the Boost software license, `http://www.boost.org/`.
- FreeImage under the FreeImage Public License v1, `http://freeimage.sourceforge.net/`.
- Qt under the GNU Lesser General Public License v2, `http://qt-project.org/`.
- zlib under the zlib license, `http://zlib.net/`.
- Qwt under the Qwt license v1, `http://qwt.sourceforge.net/`.
- Tesseract under the Apache license v2, `https://code.google.com/p/tesseract-ocr/`.
- Zbar under the GNU Lesser General Public License v2, `http://zbar.sourceforge.net/`.
- FreeType under The FreeType Project License, `http://git.savannah.gnu.org/cgit/freetype/freetype2.git/tree/docs/FTL.TXT`.

For the original terms and conditions of each of these libraries, please read the contents of the file

```
%ITWMDIR%\share\license-thirdparty.txt
```

Index