



Orfeo Toolbox

Tutorials

Mundi Copernicus DIAS
(Data and Information
Access Services)

AUTHOR

Dr. Abdelghafour HALIMI

mundiwebservices.com

Mundi Web Services, Atos, Atos Consulting, Atos Worldline, Atos Sphere, Atos Cloud and Atos WorldGrid are registered trademarks of Atos SA. July 2011. © **Copyright 2018 Mundi Web Services**

All rights reserved. Reproduction in whole or in parts is prohibited without the written consent of the copyright owner.



1. Contents

2.	The Orfeo Toolbox	3
2.1.	Introduction	3
2.2.	Application features.....	3
2.3.	Using OTB applications	3
2.3.1.	Interfaces	3
2.3.1.1.	Command line interface.....	3
2.3.1.2.	Monteverdi (Graphical user interface)	4
2.3.1.3.	Python API.....	6
3.	Tutorials	7
3.1.	Data Fusion Use Case	7
3.1.1.	Goal	7
3.1.2.	Skills acquired at the end of the training	7
3.1.3.	Training kit	7
3.1.4.	Applications.....	9
3.1.4.1.	SAR Processing	10
	• Compute Modulus And Phase.....	10
	• SAR Deburst	12
3.1.4.2.	Image Filtering	14
	• Despeckle	14
3.1.4.3.	Geometry Correction	16
	• SARMultiLook.....	16
	• OrthoRectification.....	18
3.1.4.4.	Image Manipulation	19
	• ExtractROI	19
	• Superimpose	22
	• ConcatenateImages	24
3.1.4.5.	Classification	26
	• KMeansClassification	26
3.1.5.	Summary	28
3.2.	Interferometry Use case	30
3.2.1.	Goal	30
3.2.2.	Skills acquired at the end of the training	30
3.2.3.	Training kit	30
3.2.4.	Processing Chains.....	32
3.2.4.1.	Implementation	32
3.2.4.2.	Results	35

2. The Orfeo Toolbox

2.1. Introduction

The Orfeo toolbox (OTB) is a remote sensing image processing library. The project was initiated by the CNES (French National Center for Space Studies) in 2006 and is subject to continuous development by private companies and the open-source community. The purpose of OTB is to provide users of Earth observation data the necessary tools to use these images. Originally, the library targeted the very high spatial resolution images acquired by the Orfeo constellation (Pleiades in optics, Cosmo-Skymed [CSK] in radar, but also other sensors). The OTB consists of a set of classes coded in C++ and applications built upon these classes.

2.2. Application features

OTB application features came from the C++ library classes. They meet the users' demands through simplified access and factoring requirements in data processing. They allow us, among other things, to carry out the following operations:

- Access to data: read/write most satellite image formats, access metadata, read/write vector formats (shp, kml, etc.), digital terrain models, lidar data;
- Basic image manipulation: extraction, pixel-to-pixel calculations;
- Filtering: signal processing for optical and radar imaging;
- Feature extraction: textures (Haralick, Structural Feature Set [SFS]), edge detection, points of interest, alignments, lines, descriptors (Scale-Invariant Feature Transform [SIFT], Speeded Up Robust Features [SURF]);
- Image segmentation: region growing, watershed, level set, mean-shift;
- Classification: K-means, support vector machine, random forests, as well as most recent machine learning algorithms;
- Detection of changes between images;
- Orthorectification, cartographic projections;
- Calculation of radiometric indices: vegetation, water, soil, etc.

2.3. Using OTB applications

The OTB applications aim to provide users with a number of implemented processes. These applications can be used directly through multiple interfaces (command line, GUI, Python language, etc.). Currently, there are more than 100 applications that are organized in various categories.

2.3.1. Interfaces

2.3.1.1. Command line interface

Called "Terminal" on UNIX (Linux, Mac OS X), "DOS command prompt" in Windows (cmd), this interface makes it possible to call programs without going through a graphical interface. Not very user friendly at first, this interface has many advantages, including the possibility of performing batch processes, or automating sequences of several applications. The set of executables using OTB applications on the command line are prefixed with "otbcli_".

To use an OTB application in command line, one can first get help from the application. Let us take the “Read Image Info” application for instance, which display information about the input image like: image size, origin, spacing, metadata, projections, etc. In the command prompt (on Windows) or the terminal (Linux or Mac OS X), we enter the following command for help:

```
otbcli_ReadImageInfo -help
```

This command returns the description of the application, the parameters expected as input, and also a command line example:

```
This is the ReadImageInfo application, version 7.0.0
Get information about the image
Tags: Image Manipulation Utilities Image MetaData

Display information about the input image like image size, origin, spacing, metadata,
projections...

Parameters:
MISSING -in          <string>      Input Image (mandatory)
        -keywordlist <boolean>    Display the OSSIM keywordlist (mandatory, default
        value is false)
        -outkwl      <string>      Write the OSSIM keywordlist to a geom file
        (optional, off by default)
        -rgb         <group>       Default RGB Display
        -progress    <boolean>     Report progress
        -help        <string list> Display long help (empty list), or help
        for given parameters keys

Use -help param1 [... paramN] to see detailed documentation of those parameters.

Examples:
otbcli_ReadImageInfo -in QB_Toulouse_Ortho_XS.tif
```

2.3.1.2. Monteverdi (Graphical user interface)

Monteverdi is a satellite image viewer. Its main features are:

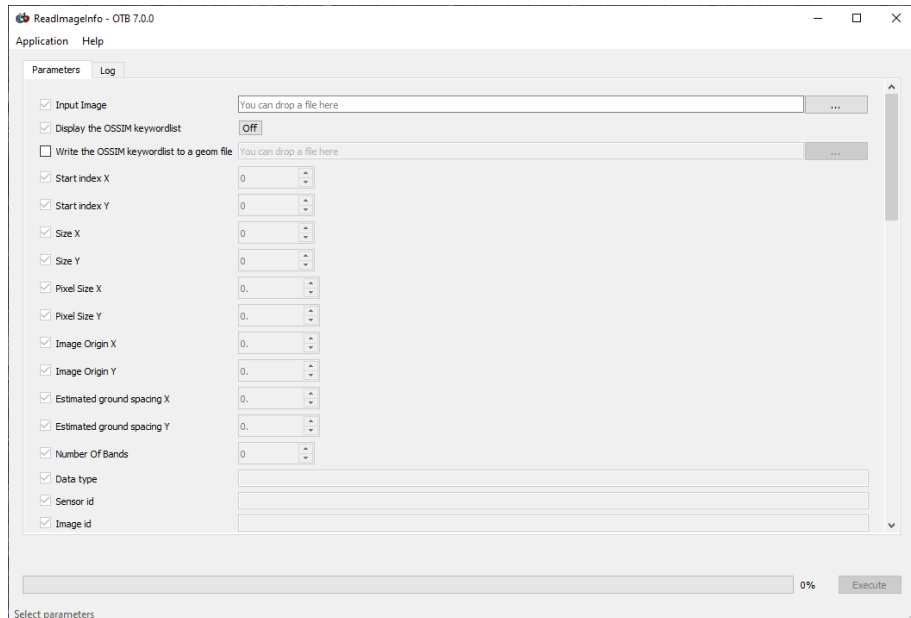
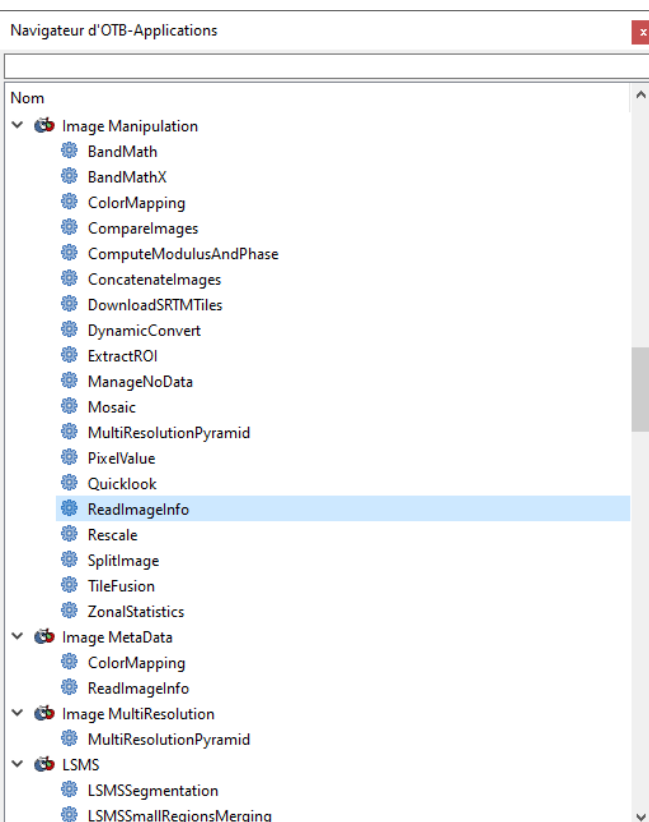
- **Performance:** Navigate instantly in full size satellite images thanks to its hardware accelerated rendering engine. Compose tiles or compare multiple images in a stack with rapid cycling and shader effects.
- **Sensor geometry support:** View raw images directly in sensor geometry! Resampling is handled by the GPU through texture mapping. OTB automagically handles coordinates mapping between actors and viewport geometries.
- **Powerful:** Access to all processing application from OTB. Orthorectification, optical calibration, classification SAR processing, and much more!



This is Monteverdi's main window where the different functionalities are:

- 1 → Main Menu
- 2 → Top toolbar
- 3 → Image View
- 4 → Widgets (OTB Application Navigator)
- 5 → Layer Stack

With the graphical user interface (GUI) of Monteverdi, it is also possible to interactively load otb-applications and use them to process images. For that purpose, the user just has to load otb-applications by clicking on the Main menu, File/Load OTB-Applications (or by simply using the shortcut CTRL+A). The figures below represent the otb-applications loading window as well as the previews example "Read Image Info" application. The applications are arranged in thematic functionalities; the user can also quickly find the wanted application by typing its name in the dedicated field at the top of the loading window.



2.3.1.3. Python API

It is possible to use OTB applications from code written in Python because of a binding mechanism. OTB applications can also be combined with Python modules for geospatial purposes.

The applications are accessed from Python, through a module named "otbApplication", an example of the "Read Image Info" application from Python is given here:

```
import otbApplication

app = otbApplication.Registry.CreateApplication("ReadImageInfo")

app.SetParameterString("in", "SLC_IW1.tif")
app.SetParameterString("outkw1", "Information.geom")

app.ExecuteAndWriteOutput()
```

3. Tutorials

3.1. Data Fusion Use Case

3.1.1. Goal

The purpose of this tutorial is to give a comprehensive overview of the ORFEO ToolBox. This tutorial also gives a better (or this tutorial allows readers to gain a better) understanding of different OTB tools (OTB application from the different interfaces) in order to process and analyze remote sensing images.

3.1.2. Skills acquired at the end of the training

At the end of the training, the reader will be able to set up OTB processing, and use OTB applications to carry out:

- Basic SAR processing
- Image manipulation
- Geometric corrections
- Orthorectification
- Data fusion
- Classification

3.1.3. Training kit

- **Mundi OTB Docker Image**

The docker version made available on Mundi, has been built by Mundi development team and can be found [here](#). Note that, you will have to log with your mundi credentials to access the mundi shared docker repository.

To use the OTB docker image on your virtual machine (VM) , docker must be installed in all Mundi VM templates and the following commands must be executed replacing `<variable>` according to your needs:

- Log yourself to the docker repository with your mundi website credentials :
`sudo docker login -u <Mundi website user email> -p <Mundi website user password> https://publicreg.mundiwebservices.com/`
- Upload the desired docker images on your VM :
`sudo docker pull publicreg.mundiwebservices.com/orfeo_toolbox:<version>`
- Launch your docker in order to use orfeo-toolbox :
`sudo docker run -it -v $(pwd):/data publicreg.mundiwebservices.com/orfeo_toolbox:<version> bash`

- **Dataset**

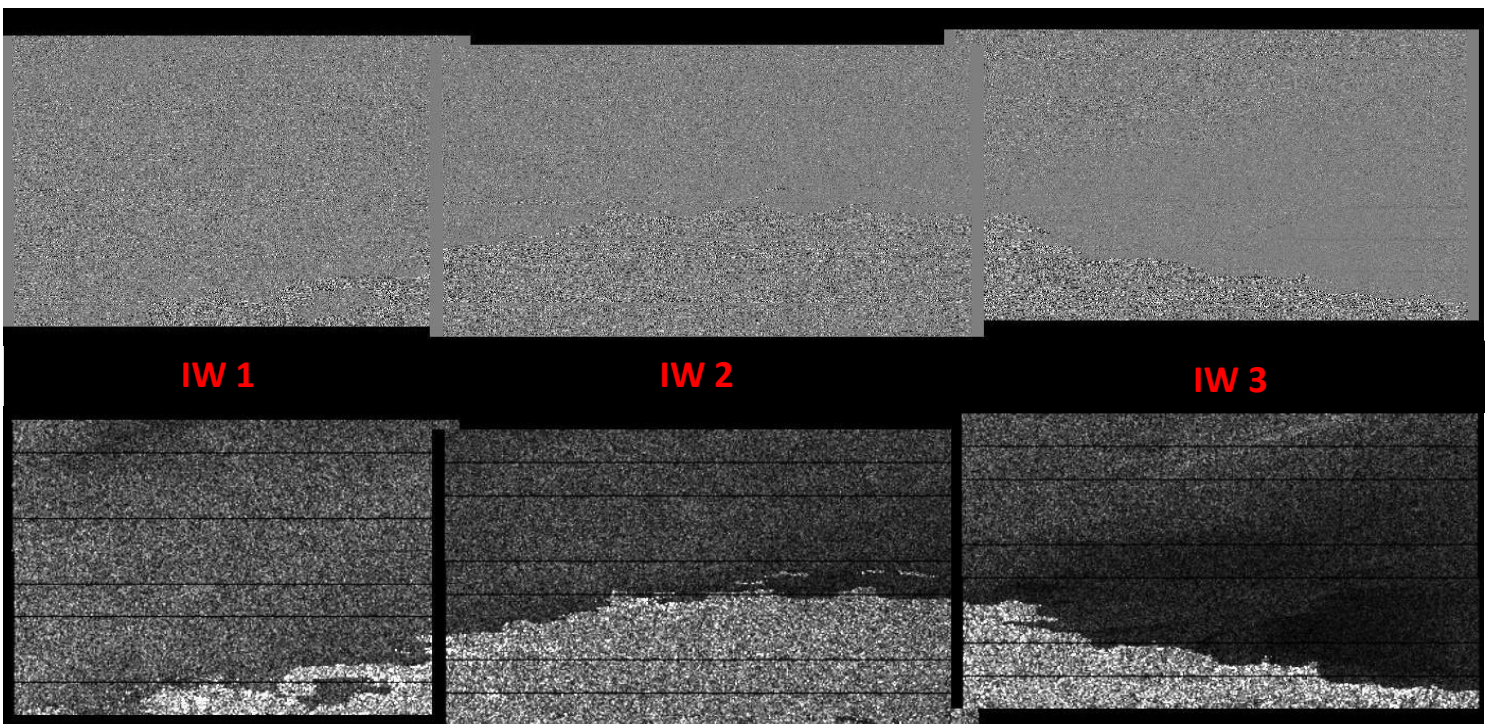
The data used in this tutorial includes images from Sentinel-1 SLC IW and Sentinel 2 L2A over the metropolitan area around Marseille, France. The figure below shows the extent and location of the Sentinel 2 data (red), the Sentinel 1 data (yellow), and the area of interest (blue).



Legend

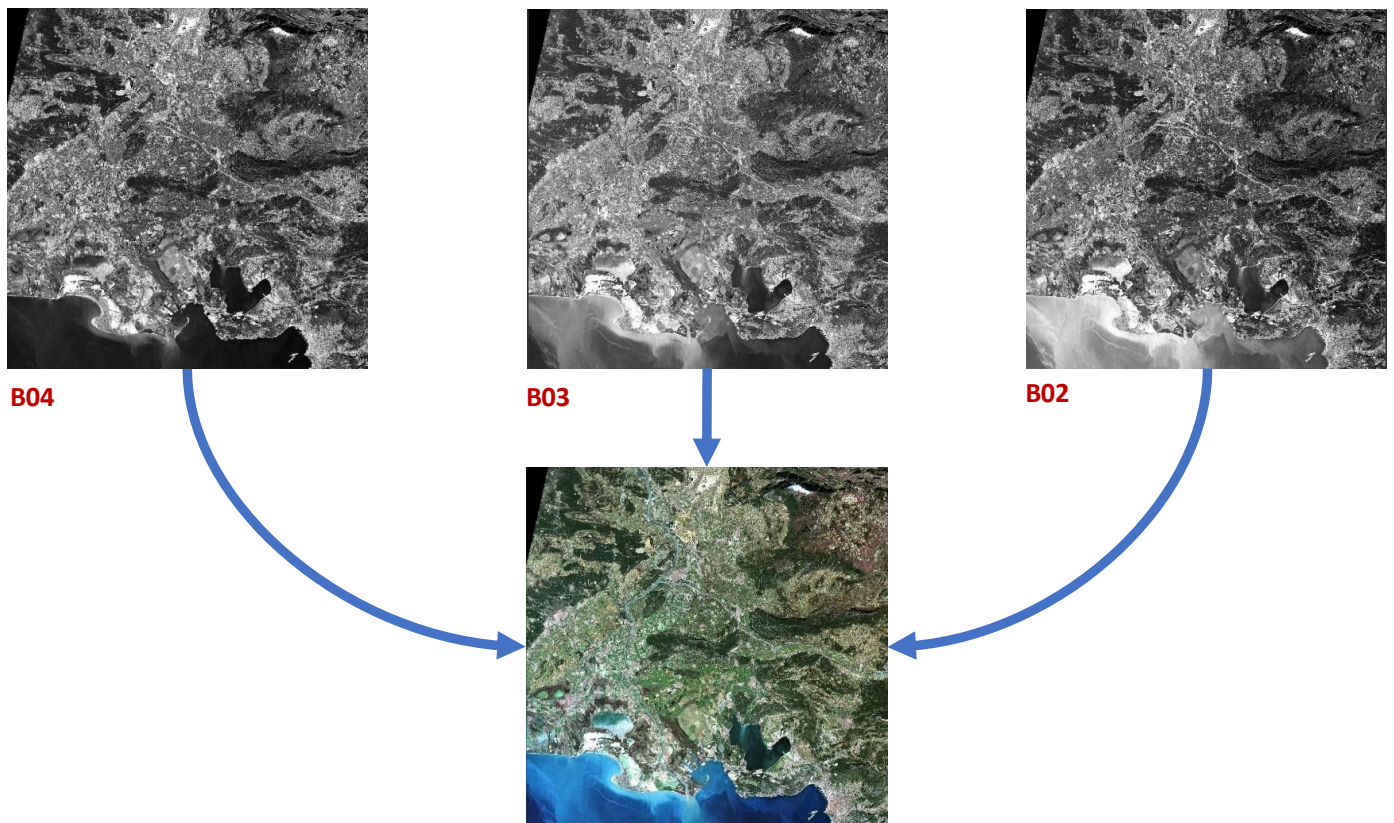
- Extent Sentinel-2 L2A data
- Extent Sentinel-1 SLC data
- Area of interest

- **Sentinel-1 SLC IW**: Single Look Complex (SLC) Interferometric Wide (IW) Swath products consist of focused SAR data geo-referenced using orbit and attitude data from the satellite and provided in zero-Doppler slant-range geometry. Mundi Web Services provides the complete collection, with fresh free data ONLINE from January 2018 with global coverage, and from January 2017 for Europe. A rolling policy of 12 months for World and 24 months for Europe is currently applied. We are making more data available every day. Discover and view the products with our Geodata UI [here](#). The IW SLC product used in this tutorial contains one image per sub-swath, per polarisation channel, for a total of three images. Each sub-swath image consists of a series of bursts, where each burst was processed as a separate SLC image. The individually focused complex burst images are included, in azimuth-time order, into a single sub-swath image, with black-fill demarcation in between as can be seen below:



- **Sentinel-2:** The Sentinel-2 L2A data is atmospherically corrected using Sen2Cor processor and PlanetDEM Digital Elevation Model (DEM). L2A products are published 48–60 hours after the LIC products and available for you to browse through our Mundi Web Services [Geodata UI](#).

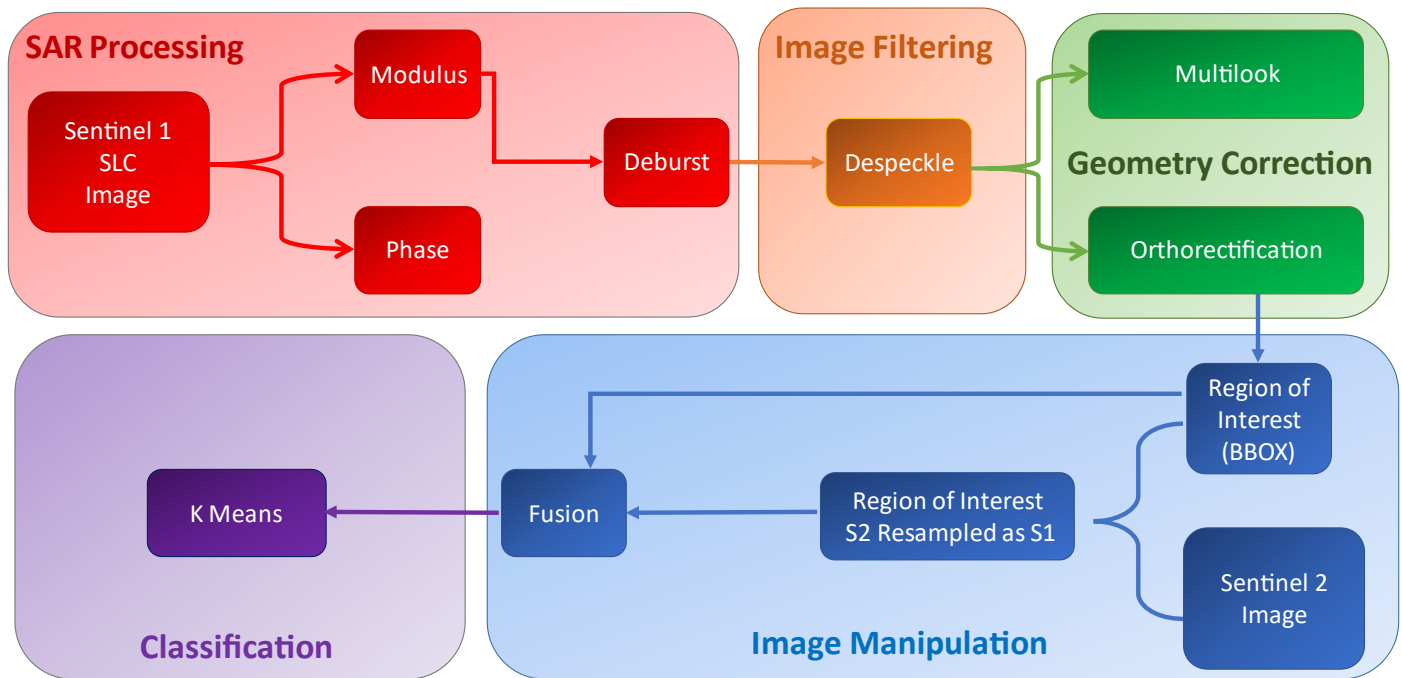
The S2-L2A product used in this tutorial consists of only three images representing the true composite color that use bands of visible light red (B04), green (B03), and blue (B02) in the corresponding red, green and blue color channels, resulting in a natural colored result. It is a good representation of the Earth as humans would see it naturally as can be seen below:



3.1.4. Applications

This section presents the use of OTB applications typically encountered in remote sensing, from image preprocessing to retrieval of information.

Each section below will introduce tools for carrying out common satellite image processing operations. In the first part, we will introduce some preprocessing tools needed to produce data that can be used in a spatial context from SLC-Sar remote sensing images. The second part will be devoted to geometric correction and orthorectification of SLC-Sar images. The third part will present some OTB applications for extracting information from the preprocessed images. Finally, applications dedicated to data fusion and classification will be presented. All these sections are organized in a pipeline depicted in the following diagram; whose components are detailed below:



3.1.4.1. SAR Processing

- **Compute Modulus And Phase**

This application computes the modulus and the phase of a complex SAR image or an image with 2 components (real and imaginary parts).

- **Description**

This application computes the modulus and the phase of a complex SAR image. The input should be a single band image with complex pixels or a 2 bands image (real and imaginary components in separate bands).

- **Parameters**

Input Image *-in image Mandatory*

Input image (complex single band or 2 bands (real/imaginary parts))

Modulus *-modulus image [dtype] Mandatory*

Modulus of the input image computes with the following formula: $\sqrt{real^2 + imag^2}$ where real and imag are respectively the real and the imaginary part of the input complex image.

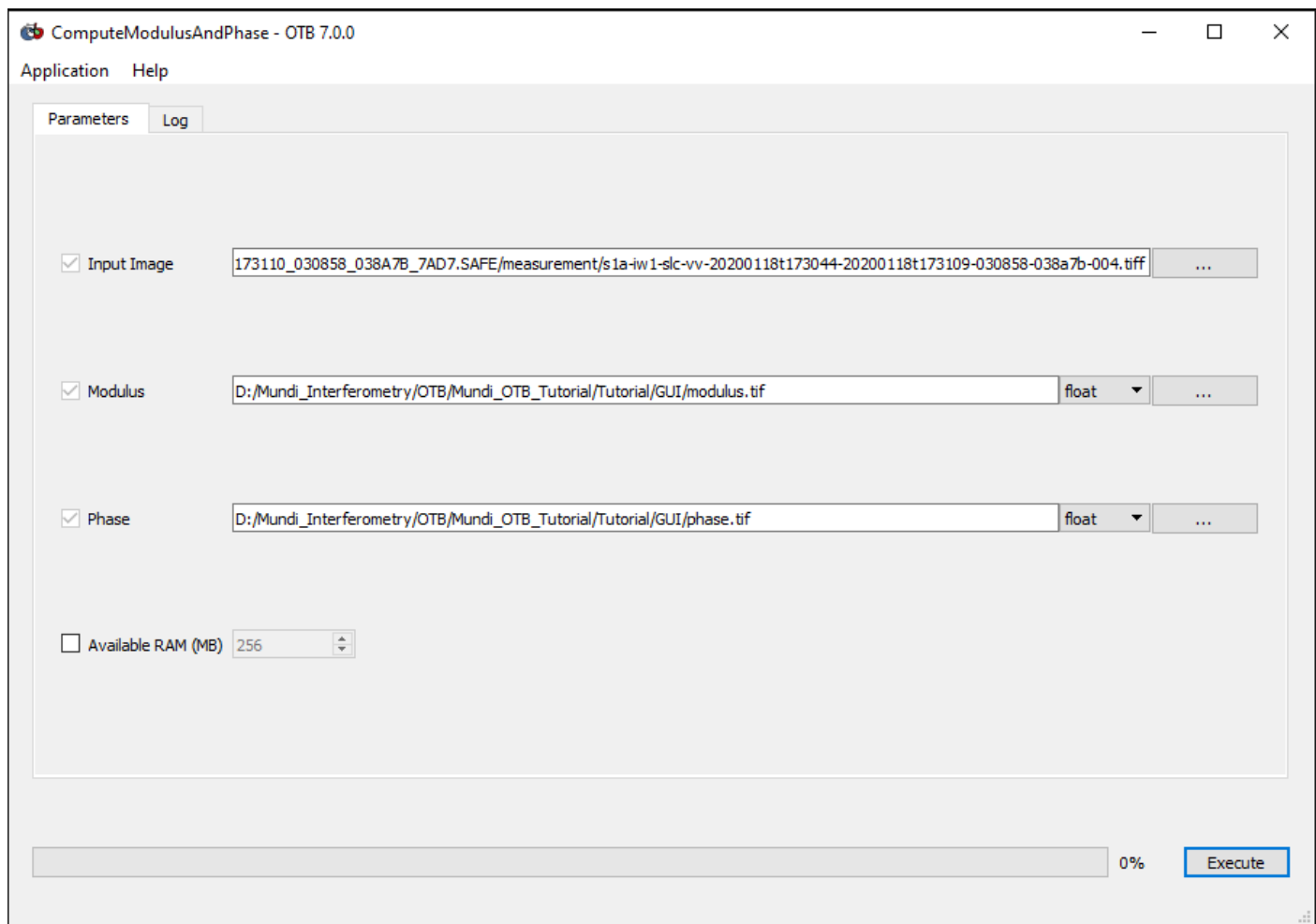
Phase *-phase image [dtype] Mandatory*

Phase of the input image computes with the following formula: $\tan^{-1}\left(\frac{imag}{real}\right)$ where real and imag are respectively the real and the imaginary part of the input complex image.

Available RAM (MB) *-ram int Default value: 256*

Available memory for processing (in MB).

- Implementations
 - Graphical User Interface



- Command line interface

```
otbcli_ComputeModulusAndPhase \
-in \
Data/S1_SLC/S1A_IW_SLC__1SDV_20200118T173043_20200118T173110_030858_038A7B_7AD7.SAFE/ \
measurement/s1a-iw1-slc-vv-20200118t173044-20200118t173109-030858-038a7b-004.tiff \
-modulus modulus.tif -phase phase.tif
```

- Python API

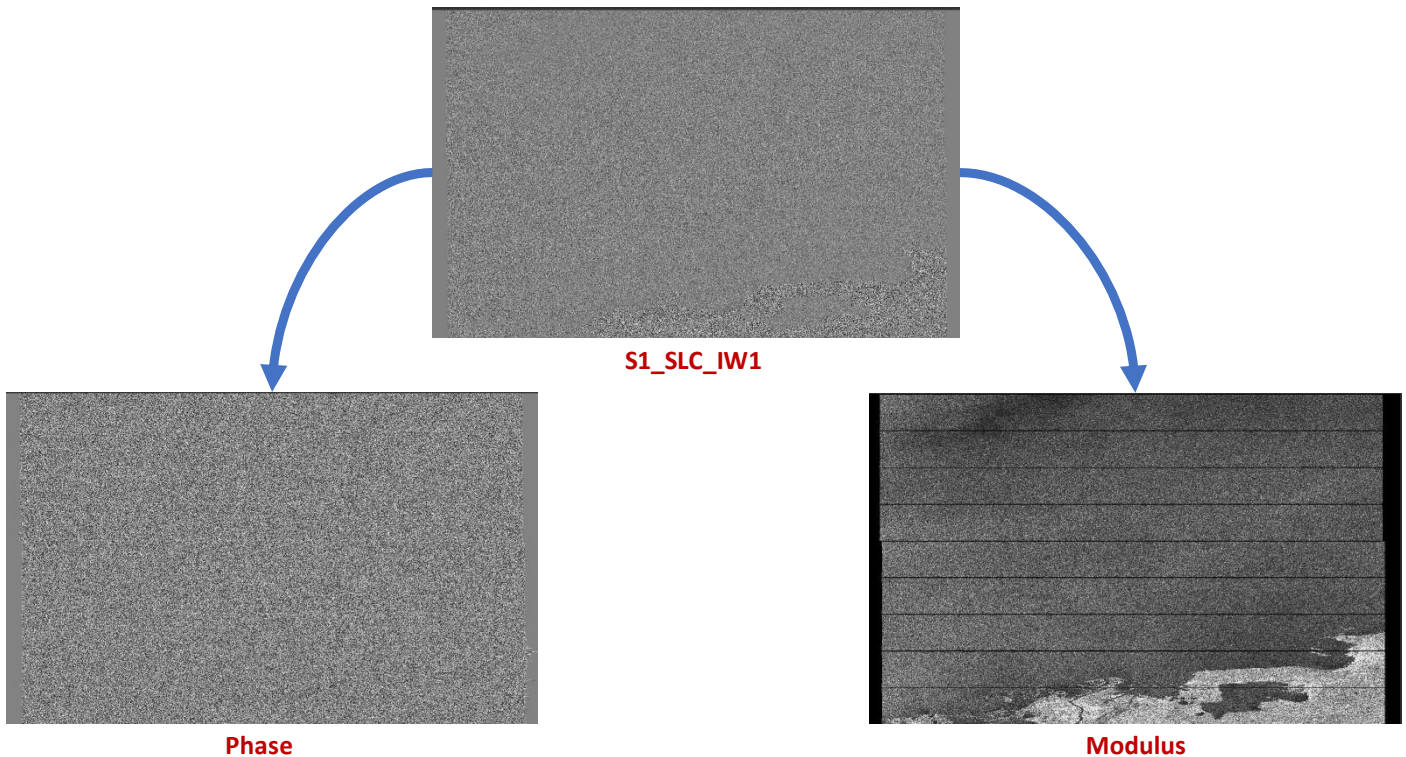
```
import otbApplication

app = otbApplication.Registry.CreateApplication("ComputeModulusAndPhase")

app.SetParameterString("in", "
Data/S1_SLC/S1A_IW_SLC__1SDV_20200118T173043_20200118T173110_030858_038A7B_7AD7.SAFE/ \
measurement/s1a-iw1-slc-vv-20200118t173044-20200118t173109-030858-038a7b-004.tiff")
app.SetParameterString("modulus", "modulus.tif")
app.SetParameterString("phase", "phase.tif")

app.ExecuteAndWriteOutput()
```


- **Result**



- **SAR Deburst**

This application performs deburst of Sentinel1 IW SLC images by removing redundant lines.

- **Description**

Sentinel1 IW SLC products are composed of several burst overlapping in azimuth time for each subswath, separated by black lines. The deburst operation consist in generating a continuous image in terms of azimuth time, by removing black separation lines as well as redundant lines between bursts.

- **Parameters**

Input Sentinel1 IW SLC Image *-in image* **Mandatory**

Raw Sentinel1 IW SLC image, or any extract of such made by OTB (geom file needed)

Output Image *-out image [dtype]* **Mandatory**

Deburst image, with updated geom file that can be further used by Orthorectification application. If the input image is a raw Sentinel1 product, uint16 output type should be used (encoding of S1 product). Otherwise, output type should match type of input image.

Select the modes for output image *-onlyvalidsamples bool* **Default value: false**

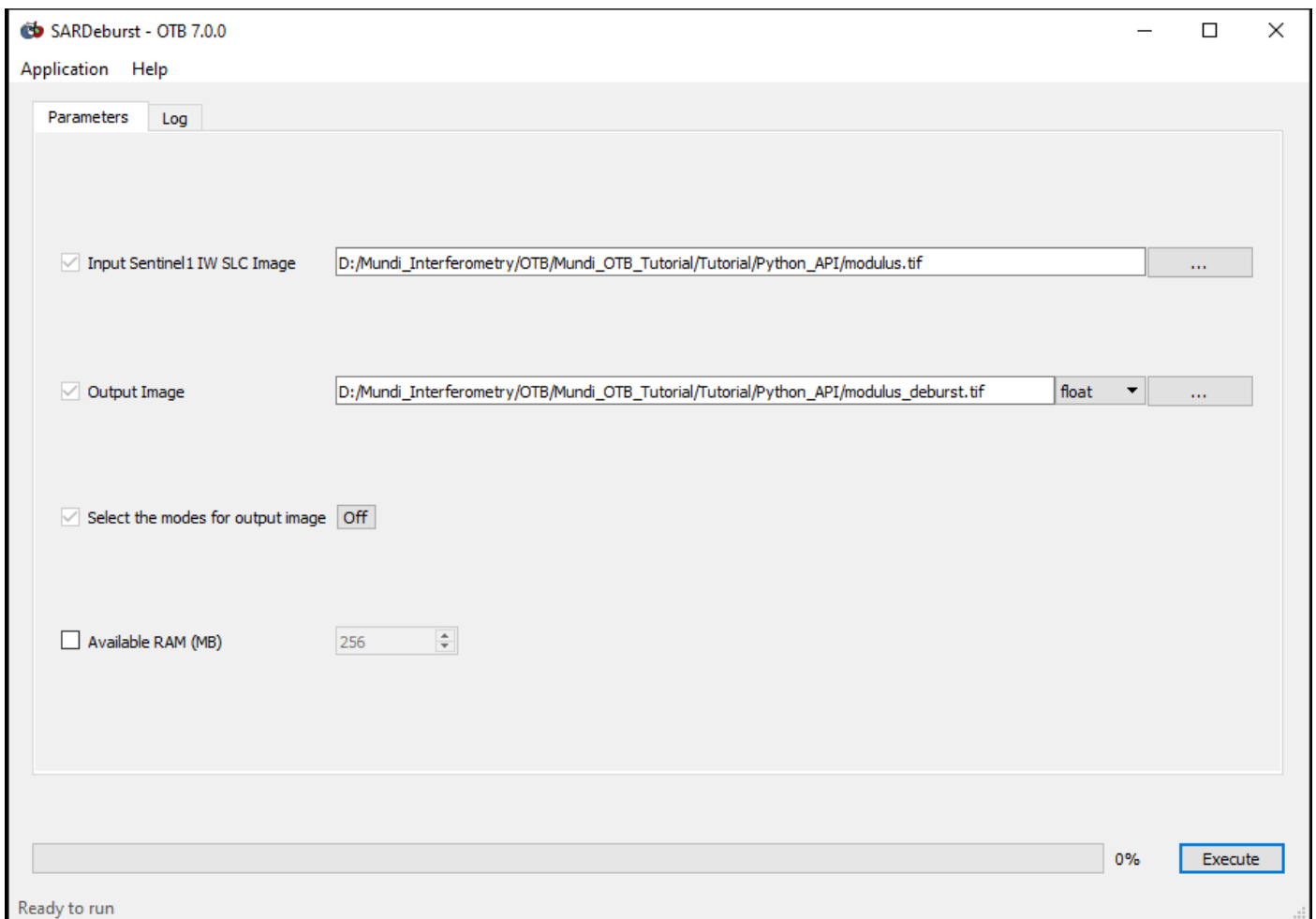
If true, the selected mode is with only valid samples.

Available RAM (MB) *-ram int* **Default value: 256**

Available memory for processing (in MB).

- **Implementations**

- **Graphical User Interface**



- **Command line interface**

```
otbcli_SARDeburst -in modulus.tif -out modulus_deburst.tif
```

- **Python API**

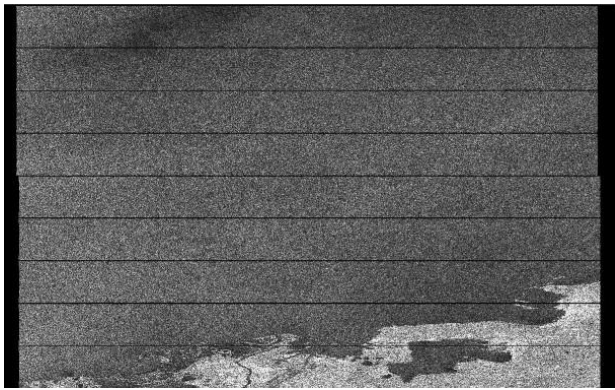
```
import otbApplication

app = otbApplication.Registry.CreateApplication("SARDeburst")

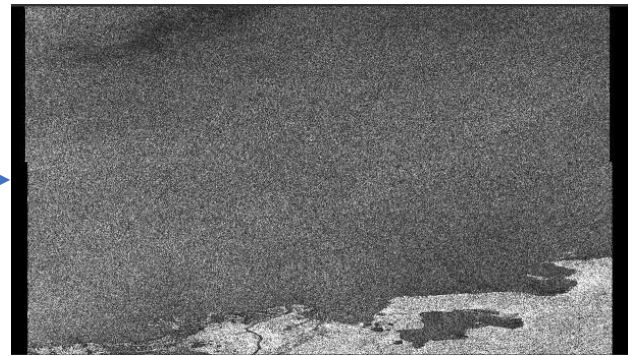
app.SetParameterString("in", "modulus.tif")
app.SetParameterString("out", "modulus_deburst.tif")

app.ExecuteAndWriteOutput()
```

- **Result**



Modulus



Modulus_Deburst

3.1.4.2. Image Filtering

- **Despeckle**

Perform speckle noise reduction on SAR image.

- **Description**

SAR images are affected by speckle noise that inherently exists in and which degrades the image quality. It is caused by the coherent nature of back-scattered waves from multiple distributed targets. It is locally strong and it increases the mean Grey level of a local area.

Reducing the speckle noise enhances radiometric resolution but tend to decrease the spatial resolution. Several different methods are used to eliminate speckle noise, based upon different mathematical models of the phenomenon. The application includes four methods: Lee, Frost , GammaMAP and Kuan.

- **Parameters**

Input Image `-in image` *Mandatory*
Input image.

Output Image `-out image [dtype]` *Mandatory*
Output image.

Speckle filtering method `-filter [lee| frost| gammamap| kuan]` *Default value: lee*

Lee options:

Radius `-filter.lee.rad int` *Default value: 1*

Radius in pixel

Number of looks `-filter.lee.nblocks float` *Default value: 1*

Number of looks in the input image.

Kuan options:

Radius `-filter.kuan.rad int` *Default value: 1*

Radius in pixel

Number of looks `-filter.kuan.nblocks float` *Default value: 1*

Number of looks in the input image.

Frost options:

Radius `-filter.frost.rad int` *Default value: 1*

Radius in pixel

Deramp factor `-filter.frost.deramp float` *Default value: 0.1*

GammaMap options:

Radius `-filter.gammamap.rad int` *Default value: 1*

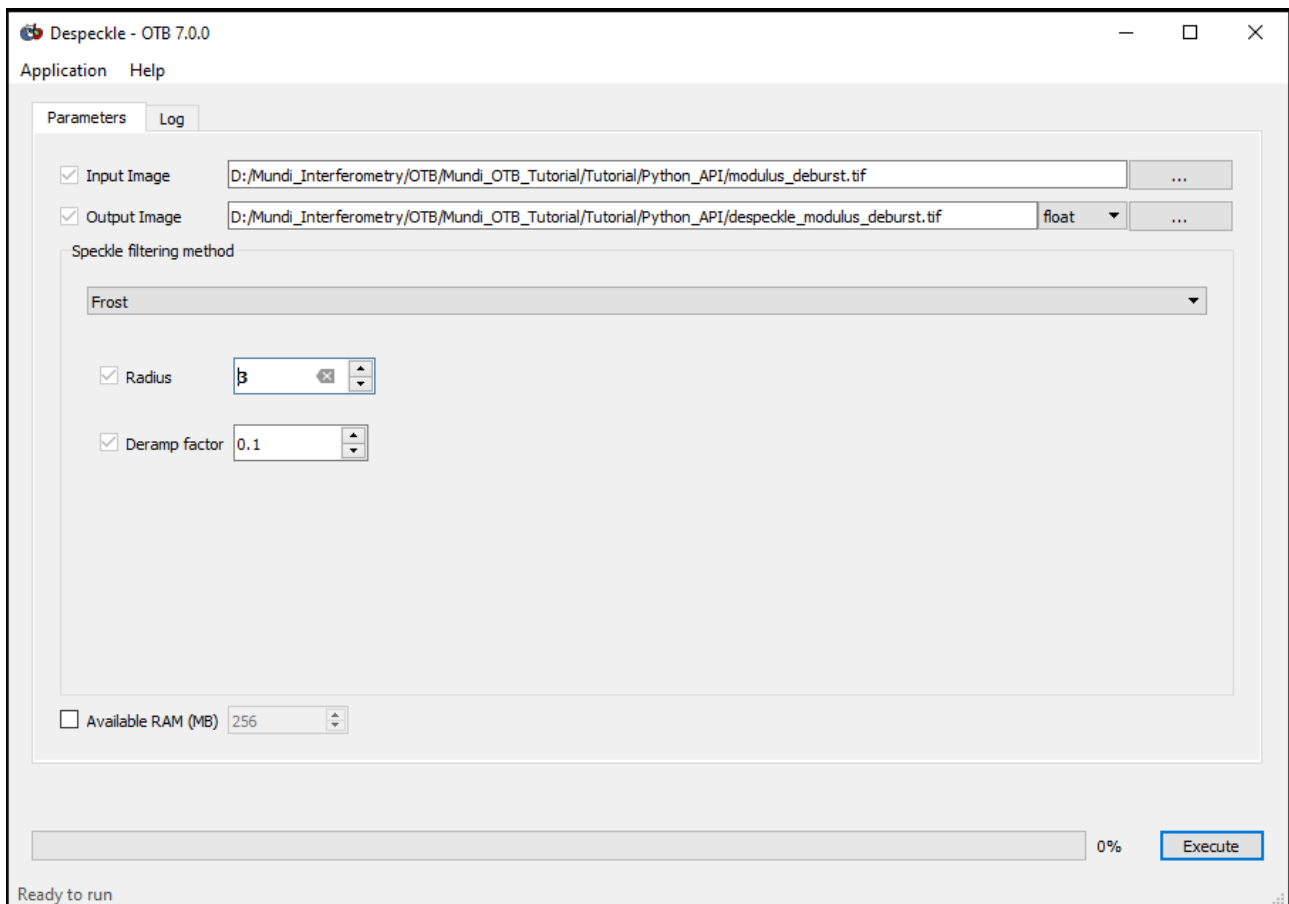
Radius in pixel

Number of looks `-filter.gammamap.nblocks float` *Default value: 1*

Number of looks in the input image.

- **Implementations**

- **Graphical User Interface**



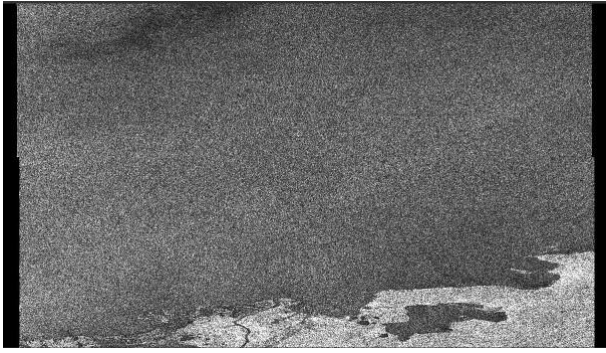
- **Command line interface**

```
otbcli_Despeckle -in modulus_deburst.tif -filter frost -filter.frost.rad 3 \  
-out despeckle_modulus_deburst.tif
```

- **Python API**

```
import otbApplication  
  
app = otbApplication.Registry.CreateApplication("Despeckle")  
  
app.SetParameterString("in", "modulus_deburst.tif")  
app.SetParameterString("filter", "frost")  
app.SetParameterInt("filter.frost.rad", 3)  
app.SetParameterString("out", "despeckle_modulus_deburst.tif")  
  
app.ExecuteAndWriteOutput()
```

- **Result**



Modulus_Deburst



Despeckle_Modulus_Deburst

3.1.4.3. Geometry Correction

- **SARMultiLook**

SAR Multi-Look creation.

- **Description**

This application creates the multi-look image of a SLC product.

- **Parameters**

Input Complex Image `-incomplex image`
Complex Image to perform computation on.

Input Image `-inreal image`
Image to perform computation on.

Averaging on distance `-mlran int` *Default value: 3*
Averaging on distance.

Averaging on azimuth `-mlazi int` *Default value: 3*
Averaging on azimuth.

Gain to apply on ML image `-mlgain float` *Default value: 0.1*
Gain to apply on ML image.

Output ML Image `-out image [dtype]` *Mandatory*
Output ML image.

Available RAM (MB) `-ram int` *Default value: 256*
Available memory for processing (in MB).

- **Implementations**

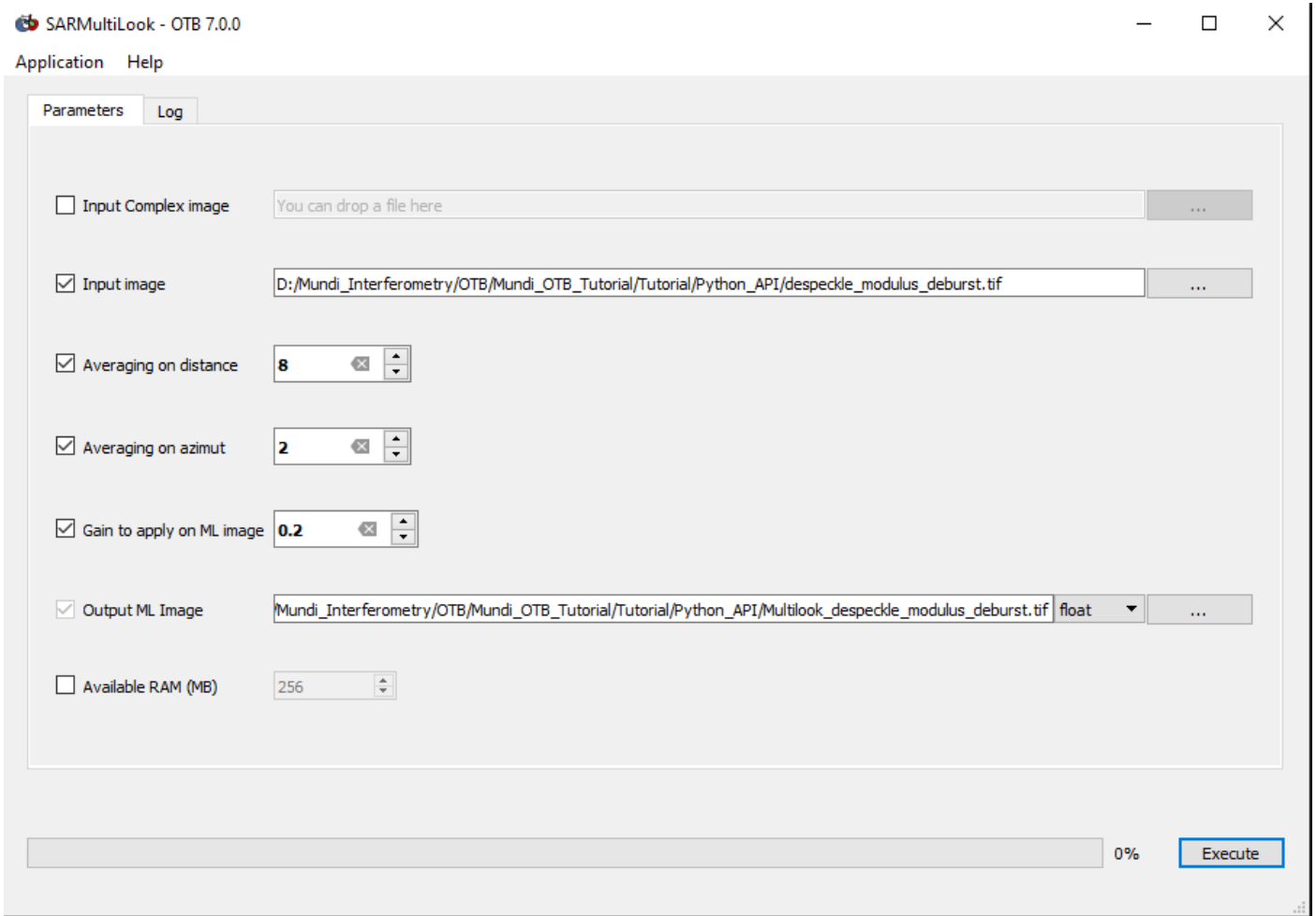
- **Command line Interface**

```
otbcli_SARMultiLook -inreal despeckle_modulus_deburst.tif \  
-out Multilook_despeckle_modulus_deburst.tif -mlran 8 -mlazi 2 -mlgain 0.2
```

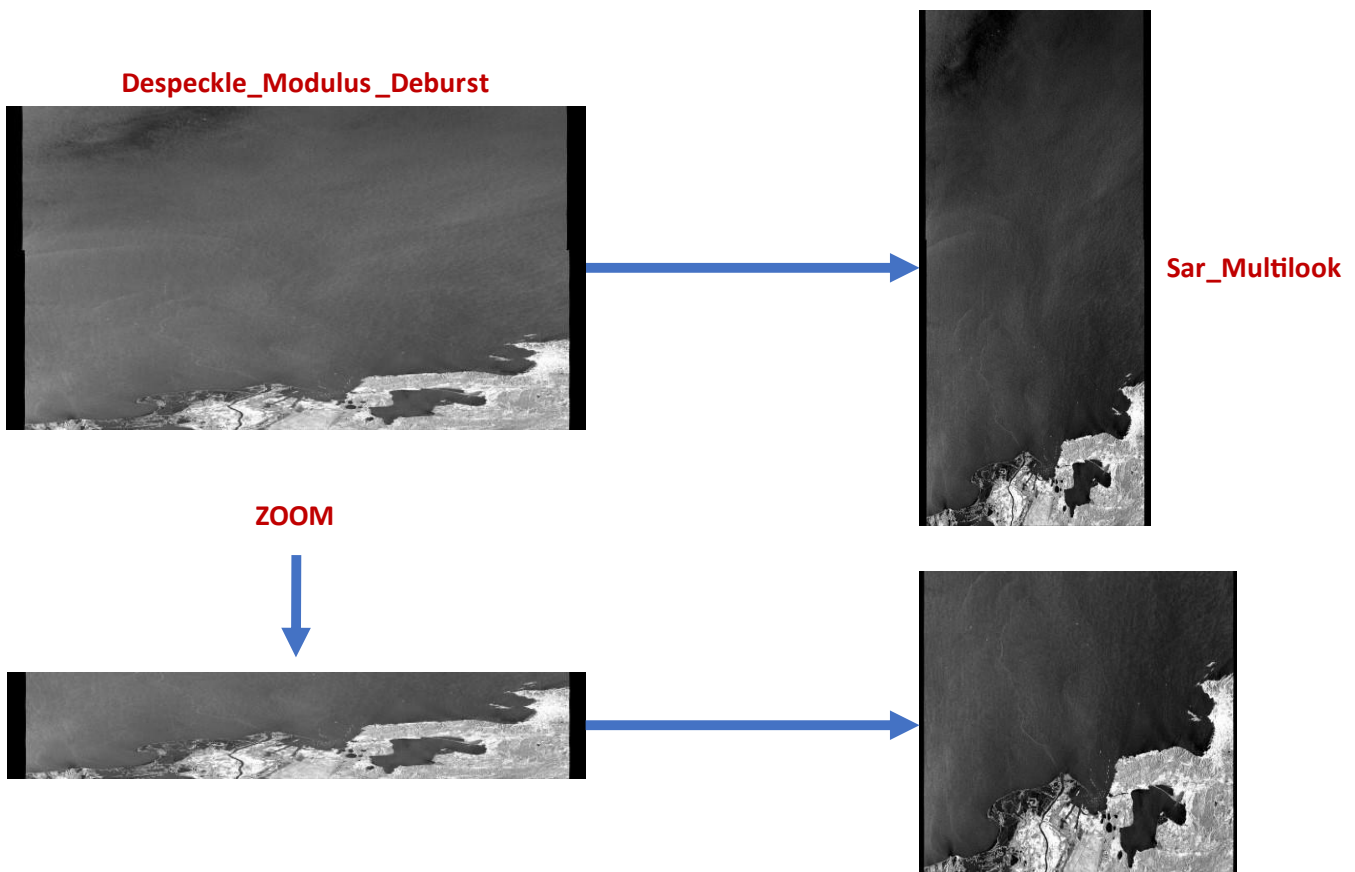
- **Python API**

```
import otbApplication  
  
app = otbApplication.Registry.CreateApplication("SARMultiLook")  
  
app.SetParameterString("inreal", "despeckle_modulus_deburst.tif")  
app.SetParameterString("out", "Multilook_despeckle_modulus_deburst.tif")  
app.SetParameterInt("mlran", 8)  
app.SetParameterInt("mlazi", 2)  
app.SetParameterInt("mlgain", 0.2)  
  
app.ExecuteAndWriteOutput()
```

- **Graphical User Interface**



- **Result**



- **OrthoRectification**

This application allows ortho-rectifying optical and radar images from supported sensors.

- **Description**

This application uses inverse sensor modelling combined with a choice of interpolation functions to resample a sensor geometry image into a ground geometry regular grid. The ground geometry regular grid is defined with respect to a map projection (see map parameter). The application offers several modes to estimate the output grid parameters (origin and ground sampling distance), including automatic estimation of image size, ground sampling distance, or both, from image metadata, user-defined ROI corners, or another ortho-image. A digital Elevation Model along with a geoid file can be specified to account for terrain deformations. In case of SPOT5 images, the sensor model can be approximated by an RPC model in order to speed-up computation.

- **Parameters**

Input Image `-io.in image` *Mandatory*
The input image to ortho-rectify

Output Image `-io.out image [dtype]` *Mandatory*
The ortho-rectified output image

DEM directory `-elev.dem directory`
This parameter allows selecting a directory containing Digital Elevation Model files.

Geoid File `-elev.geoid filename [dtype]`

Use a geoid grid to get the height above the ellipsoid in case there is no DEM available, no coverage for some points or pixels with no_data in the DEM tiles.

Default elevation `-elev.default float` *Default value: 0*

This parameter allows setting the default height above ellipsoid when there is no DEM available, no coverage for some points or pixels with no_data in the DEM tiles, and no geoid file has been set.

Map Projection `-map [utm | lambert2 | lambert93 | wgs | epsg]` *Default value: utm*
Defines the map projection to be used.

Universal Trans-Mercator (UTM) options:

Zone number `-map.utm.zone int` *Default value: 31*

Northern Hemisphere `-map.utm.northem bool` *Default value: false*

EPSG Code options:

EPSG Code `-map.epsg.code int` *Default value: 4326*

Output Image Grid

Parameters estimation modes `-outputs.mode [auto|autosize|autospacing|outputroi|orthofit]` *Default value: auto*

- **Implementations**

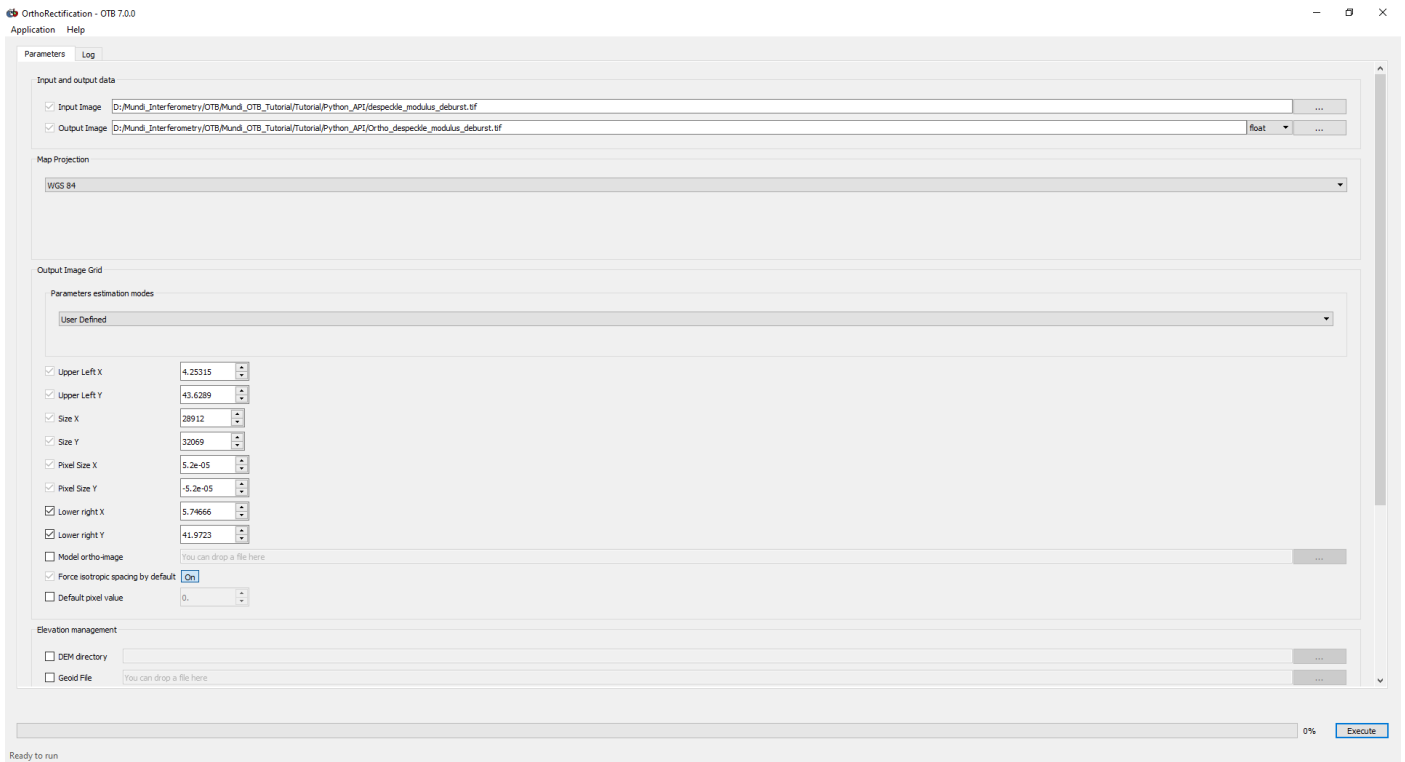
- **Command line Interface**

```
otbcli_OrthoRectification -io.in despeckle_modulus_deburst.tif \  
-io.out Ortho_despeckle_modulus_deburst.tif -map wgs
```

- **Python API**

```
import otbApplication  
  
app = otbApplication.Registry.CreateApplication("OrthoRectification")  
  
app.SetParameterString("io.in", "despeckle_modulus_deburst.tif")  
app.SetParameterString("io.out", "Ortho_despeckle_modulus_deburst.tif")  
app.SetParameterString("map", "wgs")  
  
app.ExecuteAndWriteOutput()
```

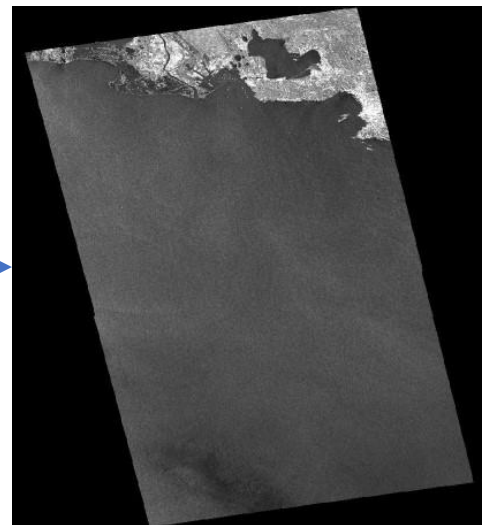
- **Graphical User Interface**



- **Result**



Despeckle_Modulus_Deburst



Orthorectified_Despeckle

3.1.4.4. Image Manipulation

- **ExtractROI**

Extract a ROI defined by the user.

- **Description**

This application extracts a Region Of Interest with user parameters. There are four mode of extraction. The standard mode allows the user to enter one point (upper left corner of the region to extract) and a size. The extent mode needs two points (upper left corner and lower right) and the radius mode need the center of the region and the radius: it will extract the rectangle containing the circle defined and limited by the image dimension. The fit mode needs a reference image or

vector and the dimension of the extracted region will be the same as the extent of the reference. Different units are available such as pixel, image physical space or longitude and latitude.

- **Parameters**

Input Image `-in image` *Mandatory*

Image to be processed

Output Image `-out image [dtype]` *Mandatory*

Region of interest from the input image

Extraction mode `-mode [standard | fit | extent | radius]`

Region of interest from the input image

Fit options:

Reference image `-mode.fit.im image` *Mandatory*

Reference image to define the ROI

Reference vector `-mode.fit.vect vectorfile` *Mandatory*

The extent of the input vector file is computed and then gives a region of interest that will be extracted.

Radius options:

Radius `mode.radius.r float` *Default value: 0*

This is the radius parameter of the radius mode.

Radius unit `-mode.radius.unitr [pxl|phy]` *Default value: pxl*

Center unit `-mode.radius.unitc [pxl|phy|lonlat]` *Default value: pxl*

Extent options:

X coordinate of the Upper left corner `-mode.extent.ulx float`

X coordinate of upper left corner point.

Y coordinate of the Upper left corner `-mode.extent.uly float`

Y coordinate of upper left corner point.

X coordinate of Lower Right corner point `-mode.extent.lrx float`

X coordinate of lower right corner point.

Y coordinate of Lower Right corner point `-mode.extent.lry float`

Y coordinate of lower right corner point.

Unit `-mode.extent.unit [pxl | phy | lonlat]` *Default value: pxl*

- **Implementations**

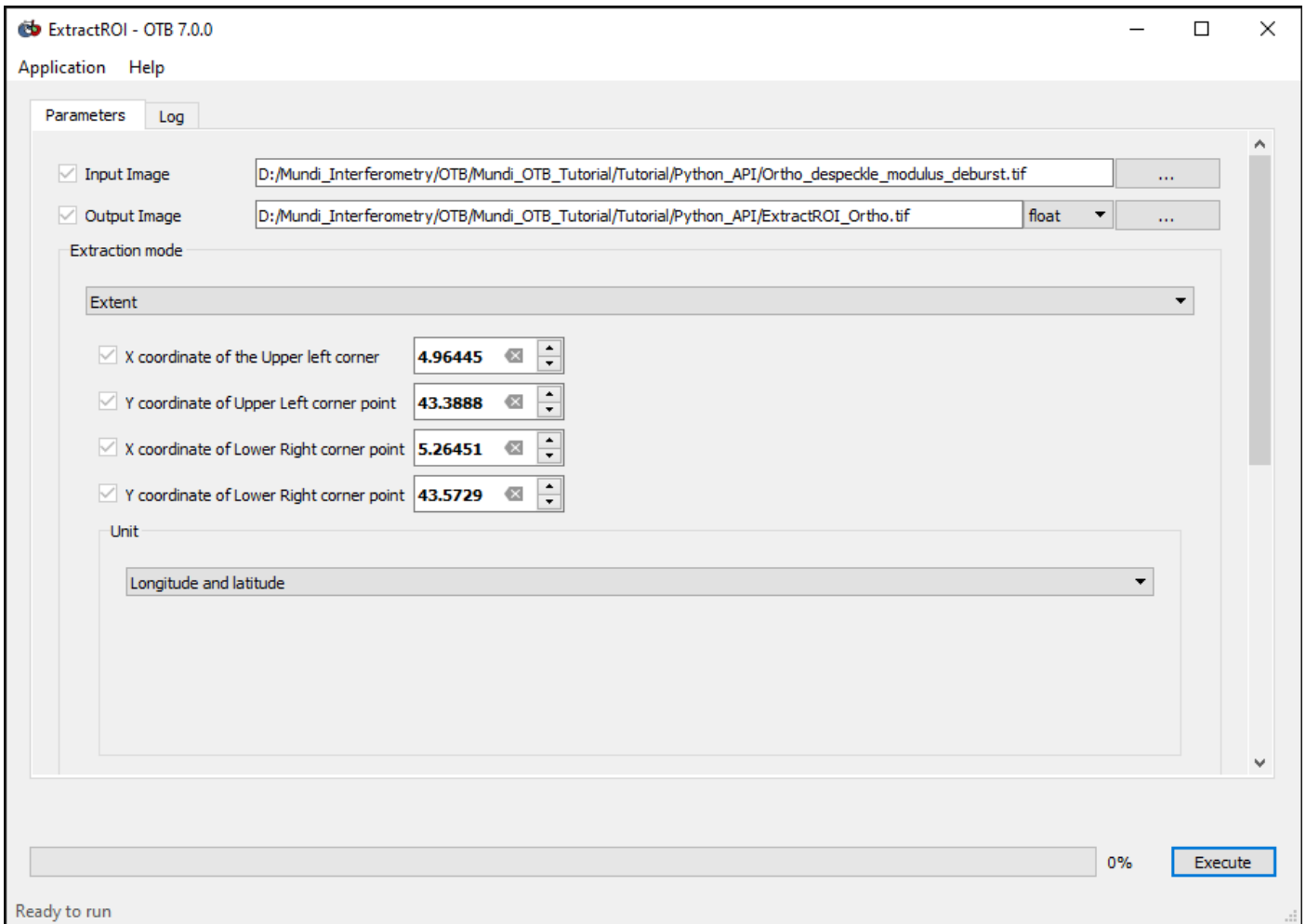
- **Command line Interface**

```
otbcli_ExtractROI -in Ortho_despeckle_modulus_deburst.tif -mode extent \  
-mode.extent.ulx 4.96445 -mode.extent.uly 43.3881 -mode.extent.lrx 5.26451 \  
-mode.extent.lry 43.5729 -mode.extent.unit lonlat -out ExtractROI_Ortho.tif
```

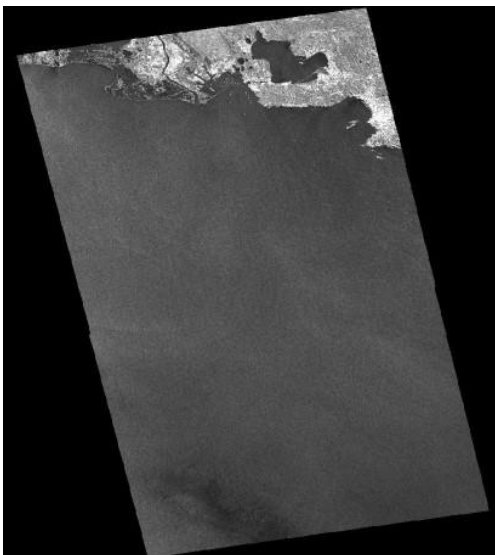
- **Python API**

```
import otbApplication  
  
app = otbApplication.Registry.CreateApplication("ExtractROI")  
  
app.SetParameterString("in", "Ortho_despeckle_modulus_deburst.tif")  
app.SetParameterString("mode", "extent")  
app.SetParameterFloat("mode.extent.ulx", 4.96445)  
app.SetParameterFloat("mode.extent.uly", 43.3881)  
app.SetParameterFloat("mode.extent.lrx", 5.26451)  
app.SetParameterFloat("mode.extent.lry", 43.5729)  
app.SetParameterString("mode.extent.unit", "lonlat")  
app.SetParameterString("out", "ExtractROI_Ortho.tif")  
  
app.ExecuteAndWriteOutput()
```

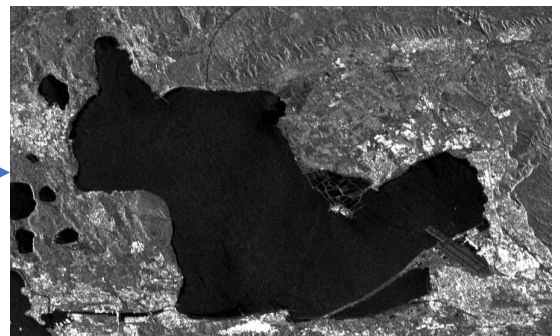
- **Graphical User Interface**



- **Result**



Orthorectified_Despeckle



Region of interest_Orthorectified

- **Superimpose**

Using available image metadata, project one image onto another one.

- **Description**

This application performs the projection of an image into the geometry of another one.

- **Parameters**

Reference input `-inr image` *Mandatory*

The input reference image.

The image to reproject `-inm image` *Mandatory*

The image to reproject into the geometry of the reference input.

Output image `-out image [dtype]` *Mandatory*

Output reprojected image.

Interpolation `-interpolator [bco | nn | linear]` *Default value: bco*

This group of parameters allows defining how the input image will be interpolated during resampling.

- **Implementations**

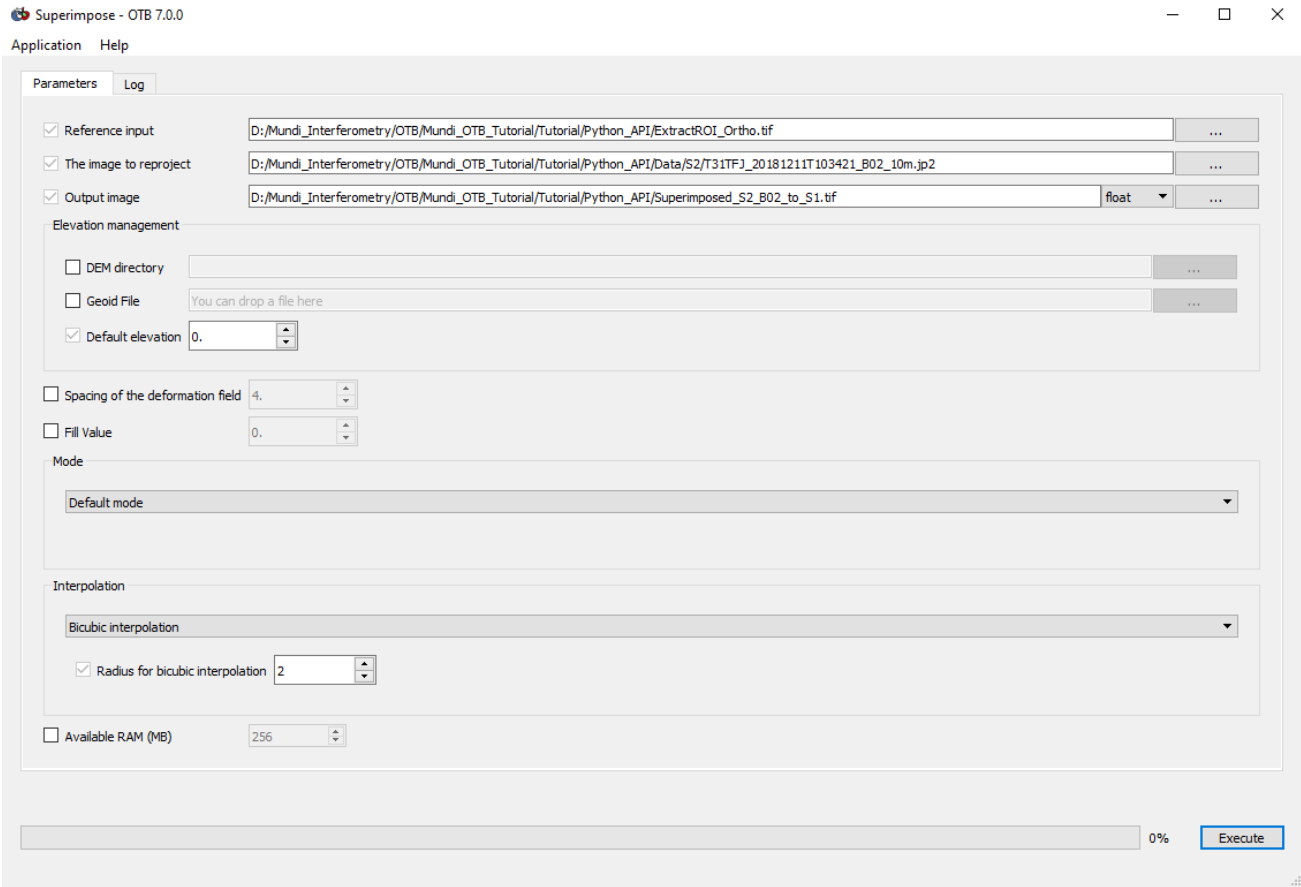
- **Command line Interface**

```
otbcli_Superimpose -inr ExtractROI_Ortho.tif -inm Data/S2/T31TFJ_20181211T103421_B02_10m.jp2 \  
-out Superimposed_S2_B02_to_S1.tif  
  
otbcli_Superimpose -inr ExtractROI_Ortho.tif -inm Data/S2/T31TFJ_20181211T103421_B03_10m.jp2 \  
-out Superimposed_S2_B03_to_S1.tif  
  
otbcli_Superimpose -inr ExtractROI_Ortho.tif -inm Data/S2/T31TFJ_20181211T103421_B04_10m.jp2 \  
-out Superimposed_S2_B04_to_S1.tif
```

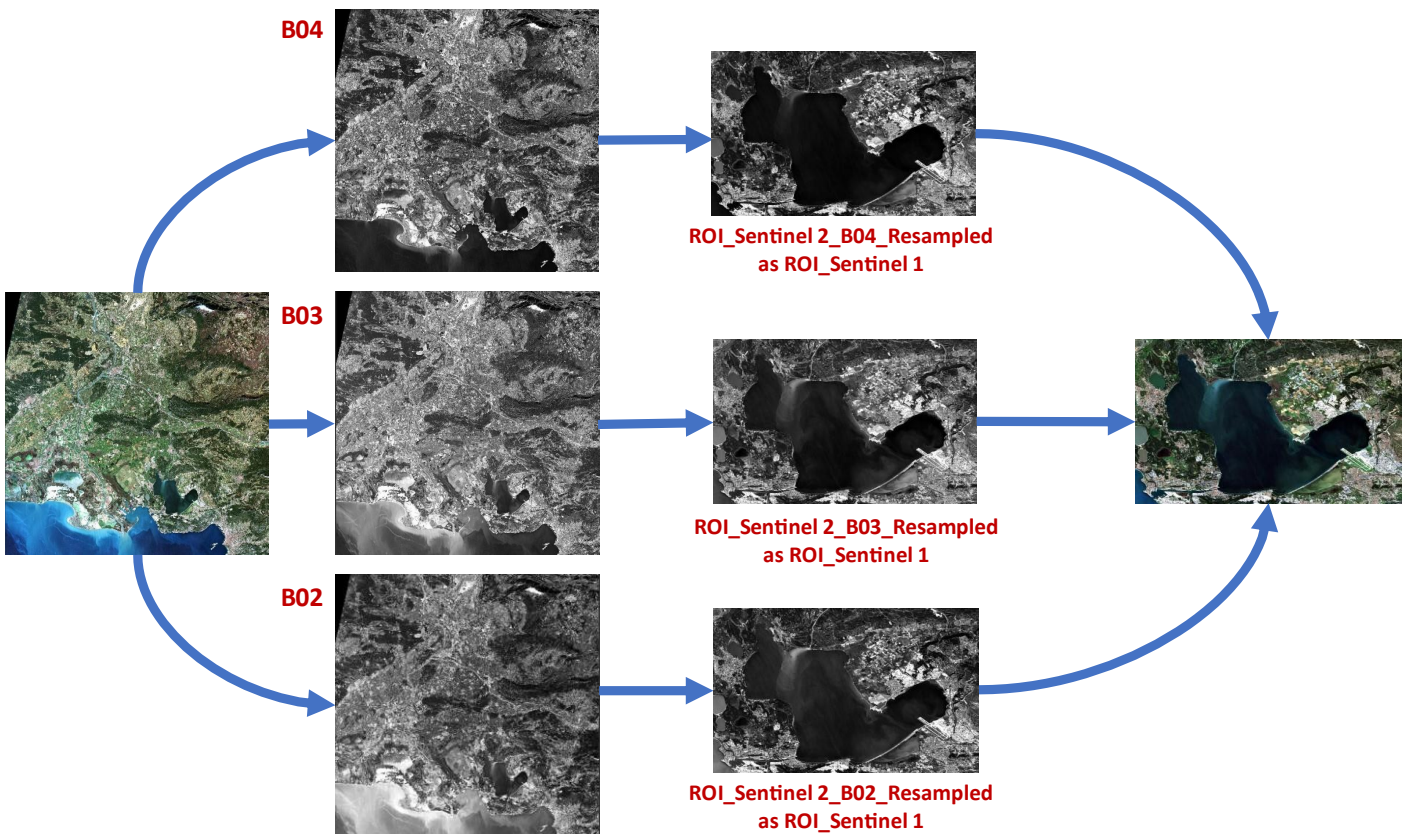
- **Python API**

```
import otbApplication  
  
app = otbApplication.Registry.CreateApplication("Superimpose")  
  
app.SetParameterString("inr", "ExtractROI_Ortho.tif")  
app.SetParameterString("inm", "Data/S2/T31TFJ_20181211T103421_B02_10m.jp2")  
app.SetParameterString("out", "Superimposed_S2_B02_to_S1.tif")  
  
app.SetParameterString("inr", "ExtractROI_Ortho.tif")  
app.SetParameterString("inm", "Data/S2/T31TFJ_20181211T103421_B03_10m.jp2")  
app.SetParameterString("out", "Superimposed_S2_B03_to_S1.tif")  
  
app.SetParameterString("inr", "ExtractROI_Ortho.tif")  
app.SetParameterString("inm", "Data/S2/T31TFJ_20181211T103421_B04_10m.jp2")  
app.SetParameterString("out", "Superimposed_S2_B04_to_S1.tif")  
  
app.ExecuteAndWriteOutput()
```

• Graphical User Interface



• Result



- **ConcatenateImages**

Concatenate a list of images of the same size into a single multi-channel image.

- **Description**

Concatenate a list of images of the same size into a single multi-channel image. It reads the input image list (single or multi-channel) and generates a single multi-channel image. The channel order is the same as the list.

- **Parameters**

Input images list *-il image1 image2... Mandatory*

The list of images to concatenate, must have the same size.

Output Image *-out image [dtype] Mandatory*

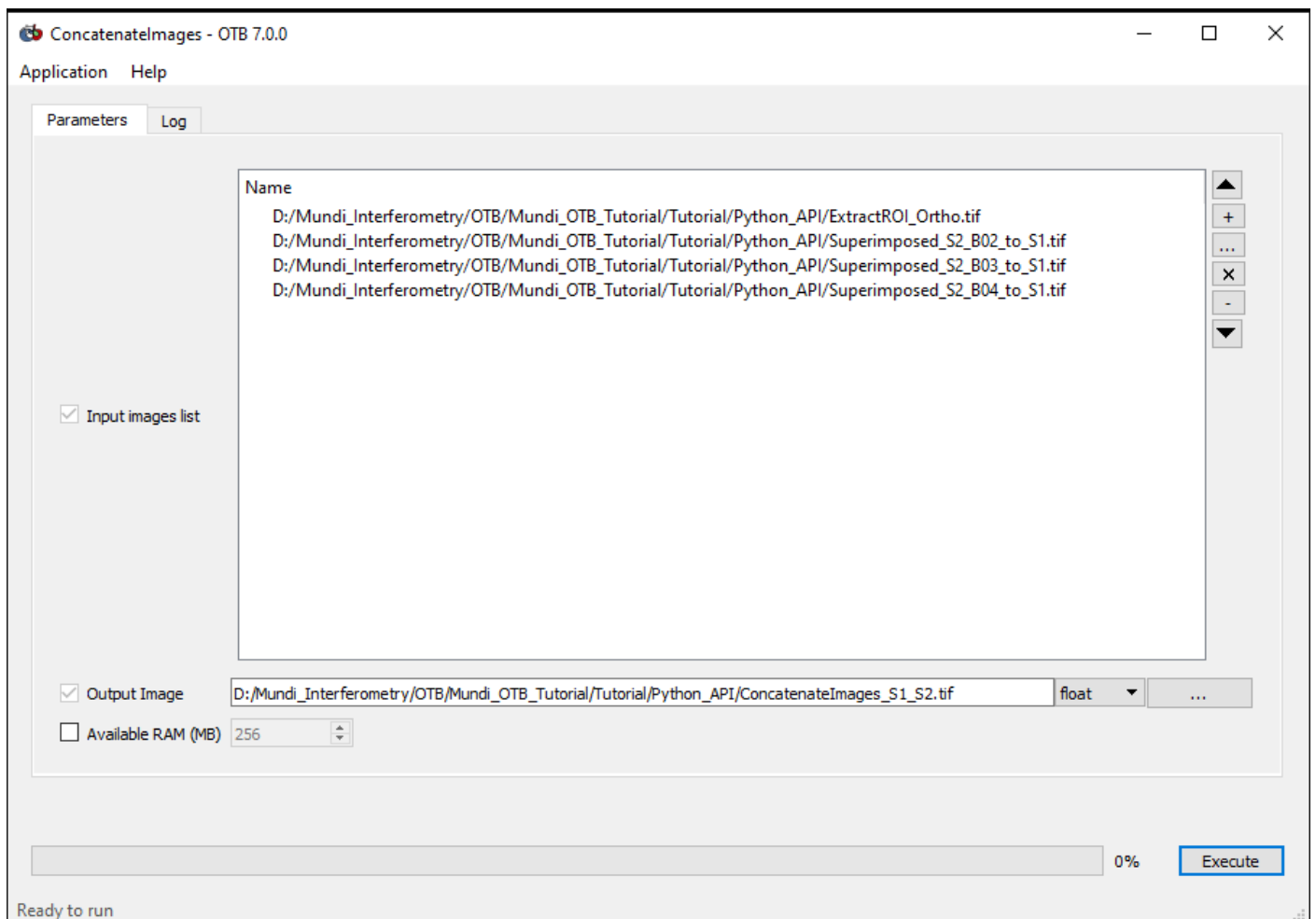
The concatenated output image.

Available RAM (MB) *-ram int Default value: 256*

Available memory for processing (in MB).

- **Implementations**

- **Graphical User Interface**



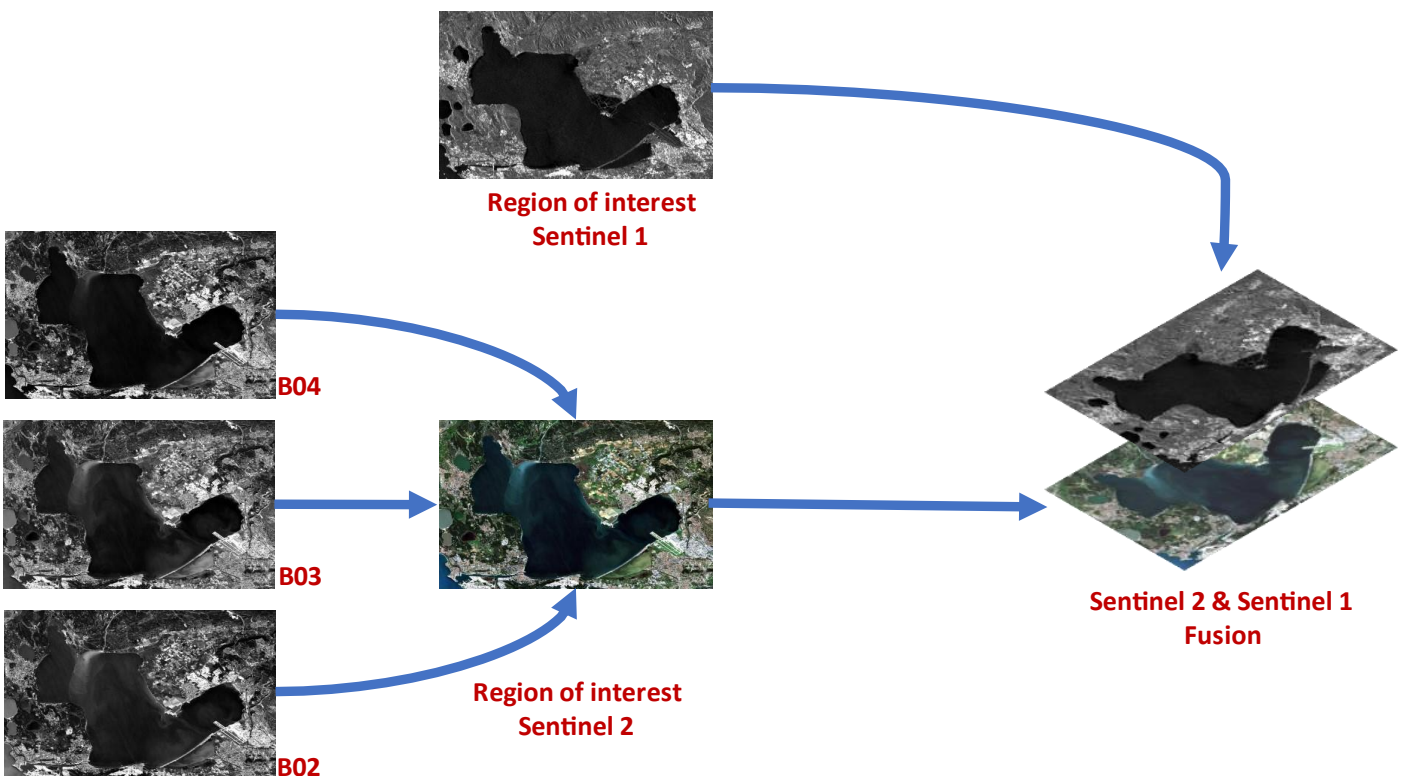
- **Command line Interface**

```
otbcli_ConcatenateImages -il ExtractROI_Ortho.tif Superimposed_S2_B02_to_S1.tif \  
Superimposed_S2_B03_to_S1.tif Superimposed_S2_B04_to_S1.tif Superimposed_S2_B08_to_S1.tif \  
-out ConcatenateImages_S1_S2.tif
```

- **Python API**

```
import otbApplication  
  
app = otbApplication.Registry.CreateApplication("ConcatenateImages")  
  
app.SetParameterStringList("il", ['ExtractROI_Ortho.tif', \  
'Superimposed_S2_B02_to_S1.tif', 'Superimposed_S2_B03_to_S1.tif', \  
'Superimposed_S2_B04_to_S1.tif', 'Superimposed_S2_B08_to_S1.tif'])  
app.SetParameterString("out", "ConcatenateImages_S1_S2.tif")  
  
app.ExecuteAndWriteOutput()
```

- **Result**



3.1.4.5. Classification

- **KMeansClassification**

Unsupervised KMeans image classification

- **Description**

Unsupervised KMeans image classification. This is a composite application, using existing training and classification applications. The SharkKMeans model is used.

- **Parameters**

Input Image `-in image` *Mandatory*

Input image filename.

Output Image `-out image [dtype]` *Mandatory*

Output image containing class labels

Number of classes `-nc int` *Default value: 5*

Number of modes, which will be used to generate class membership.

Training set size `-ts int` *Default value: 100*

Size of the training set (in pixels).

Maximum number of iterations `-maxit int` *Default value: 1000*

Maximum number of iterations for the learning step.

If this parameter is set to 0, the KMeans algorithm will not stop until convergence

- **Implementations**

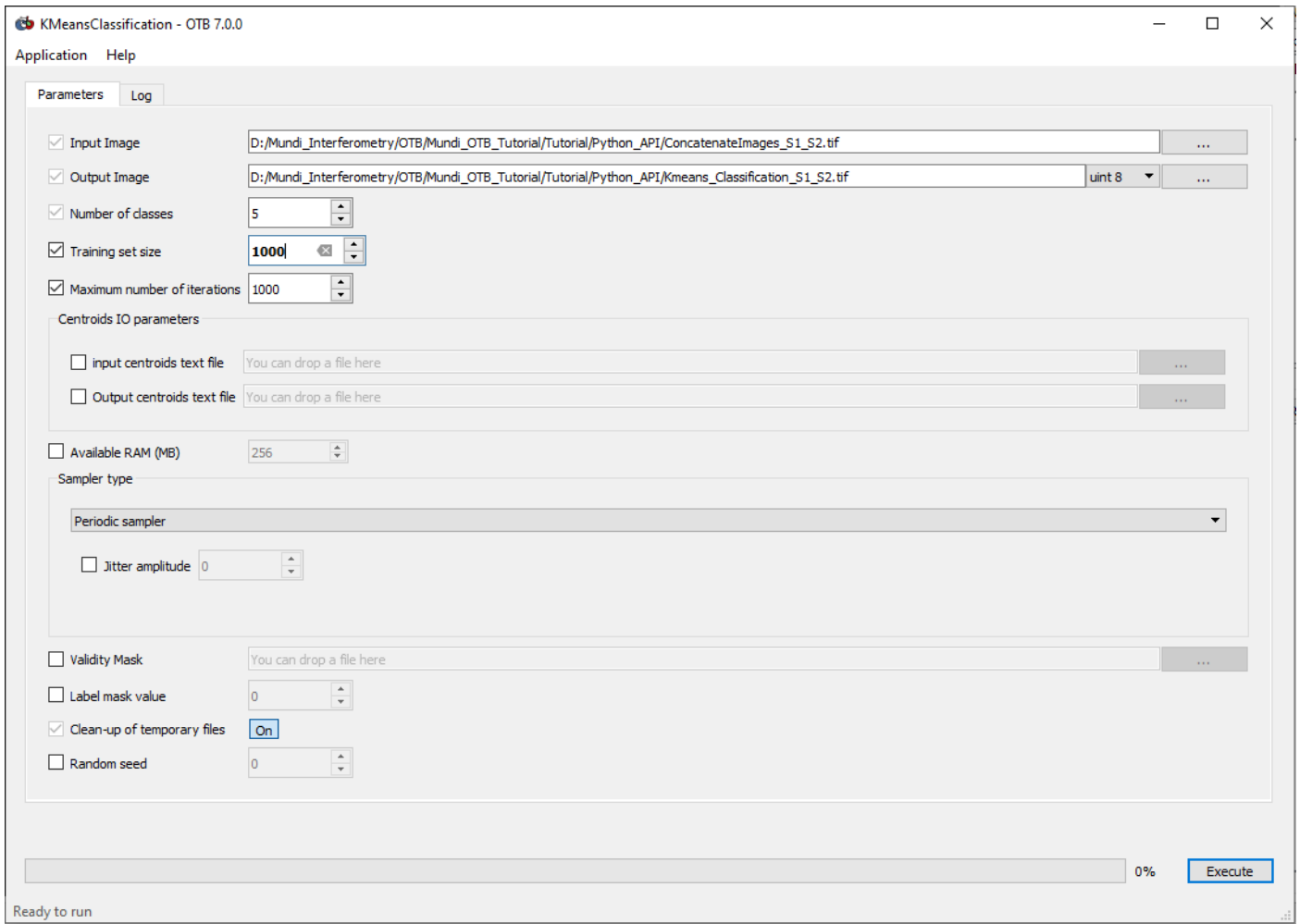
- **Command line Interface**

```
otbcli_KMeansClassification -in ConcatenateImages_S1_S2.tif -ts 1000 -nc 5 -maxit 1000 \  
-out Kmeans_Classification_S1_S2.tif uint8
```

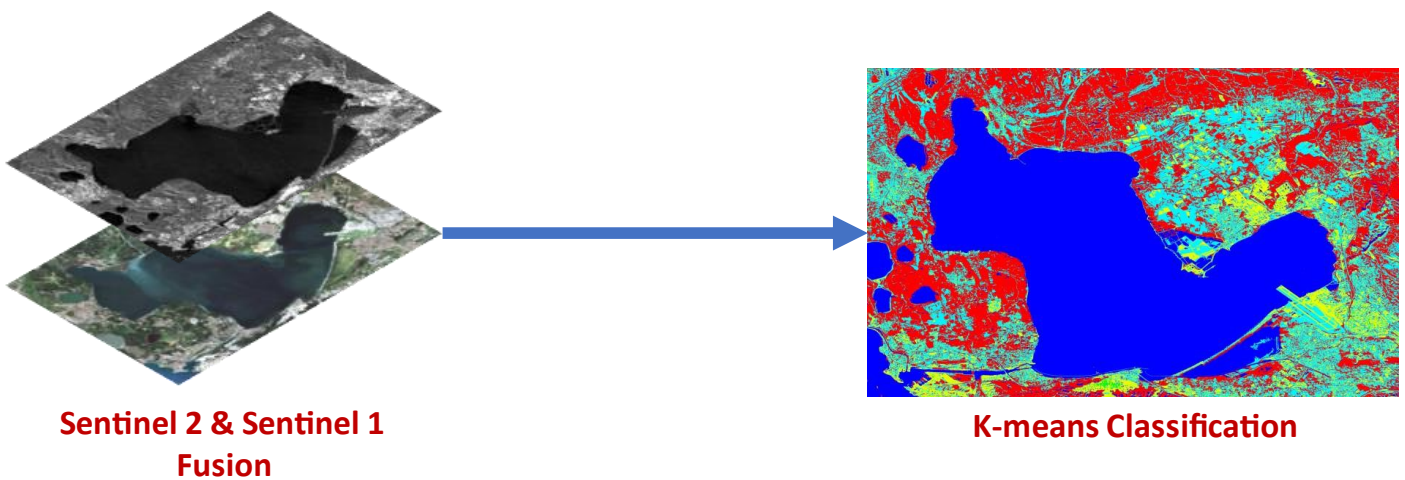
- **Python API**

```
import otbApplication  
  
app = otbApplication.Registry.CreateApplication("KMeansClassification")  
  
app.SetParameterString("in", "ConcatenateImages_S1_S2.tif")  
app.SetParameterInt("ts", 1000)  
app.SetParameterInt("nc", 5)  
app.SetParameterInt("maxit", 1000)  
app.SetParameterString("out", "Kmeans_Classification_S1_S2.tif")  
app.SetParameterOutputImagePixelFormat("out", 1)  
  
app.ExecuteAndWriteOutput()
```

- **Graphical User Interface**

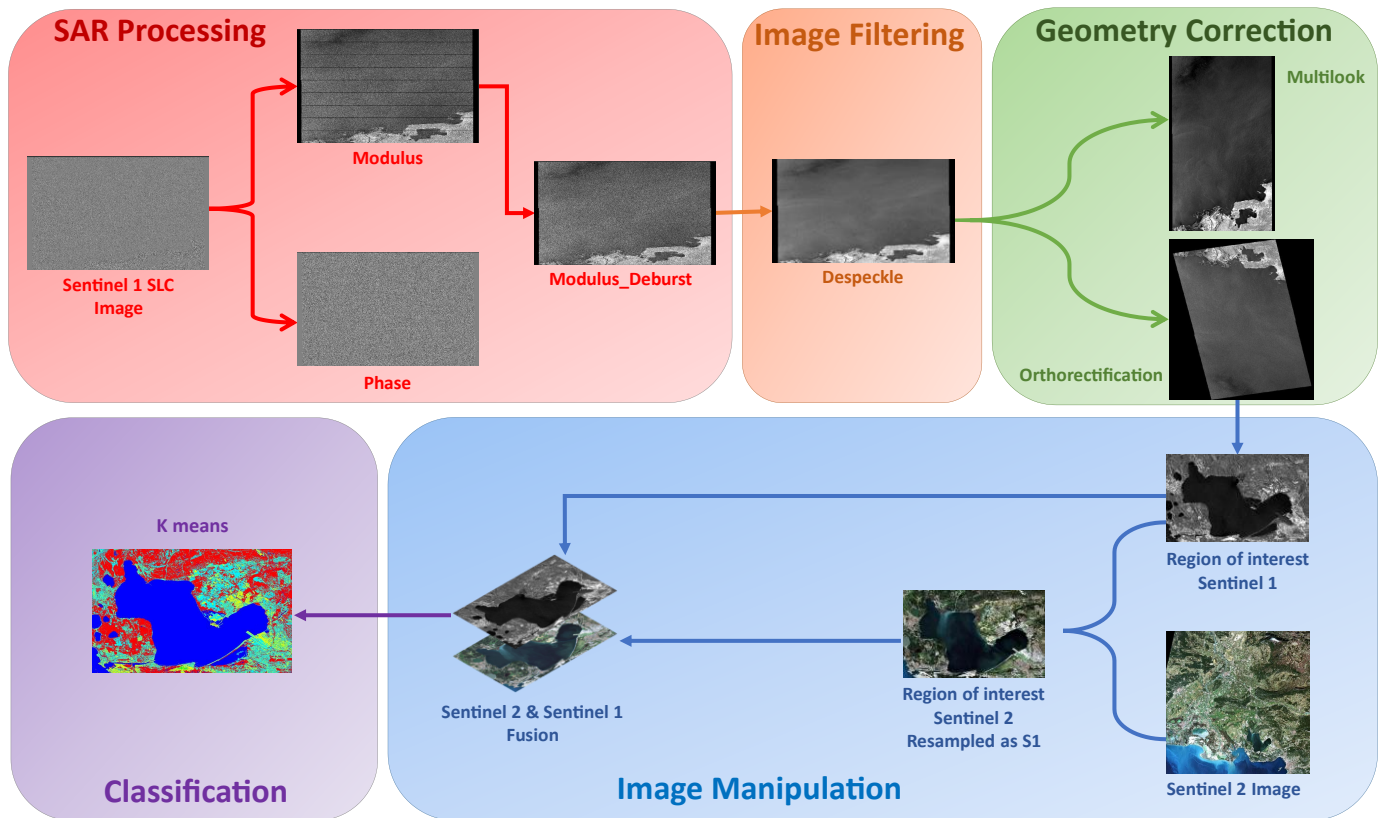


- **Result**

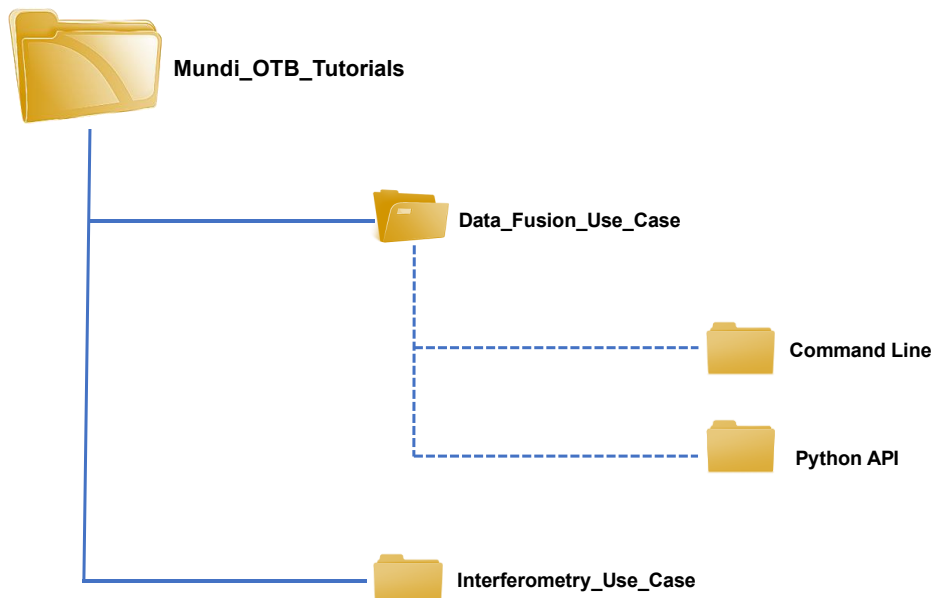


3.1.5. Summary

All the results obtained previously are summarized in the following diagram:

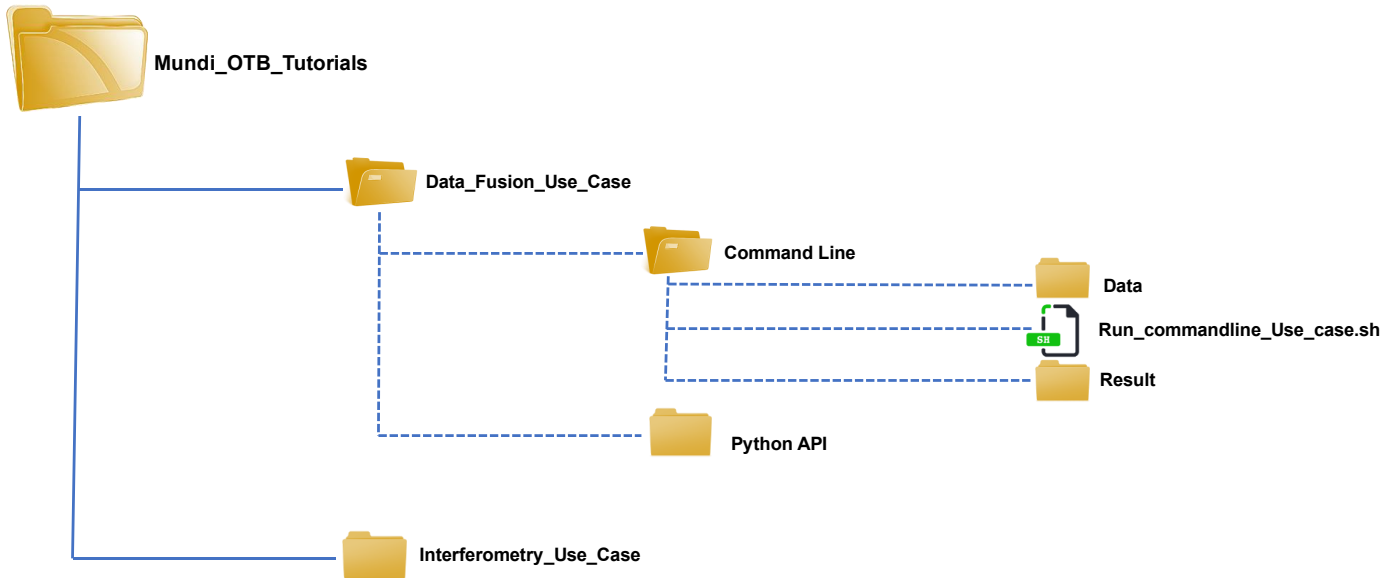


Mundi Web Services provides you with a script that allows you to run all the process in one step, whether with the command line interface or with the python API. The different ways to run this process can be found in the Mundi tutorial folder (precisely data_fusion_use_case folder) and are organized as follows:



- **Command line**

In order to run the process using the command line script you have to move to the command line directory like this

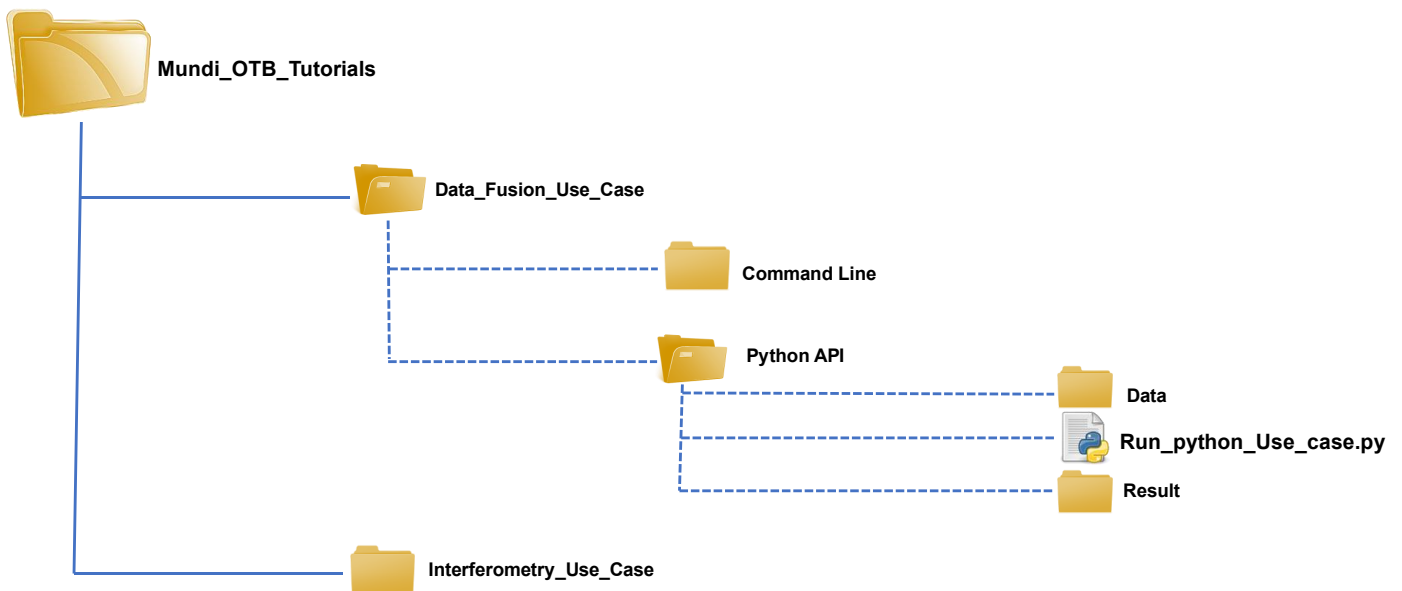


And then run the following command

```
sudo docker run -it -v $(pwd):/Tutorial -w /Tutorial publicreg.mundiwebservices.com/orfeo_toolbox:<version> bash Run_commandline_Use_case
```

- **Python API**

Same thing for the python script you have to move to the Python API directory like this



And then run the following command

```
sudo docker run -it -v $(pwd):/Tutorial -w /Tutorial publicreg.mundiwebservices.com/orfeo_toolbox:<version> python3 Run_python_Use_case.py
```


3.2. Interferometry Use case

What is Interferometry?

Interferometric synthetic aperture radar (InSAR) exploits the phase difference between two complex radar SAR observations taken from slightly different sensor positions and extracts information about the earth's surface.

A SAR signal contains amplitude and phase information. The amplitude is the strength of the radar response and the phase is the fraction of one complete sine wave cycle (a single SAR wavelength). The phase of the SAR image is determined primarily by the distance between the satellite antenna and the ground targets. By combining the phase of these two images after coregistration, an interferogram can be generated whose phase is highly correlated to the terrain topography. In the case of differential interferometry (DInSAR), this topographic phase contribution is removed using a digital elevation model (DEM). The remaining variation in the interferogram can be attributed to surface changes which occurred between the two image acquisition dates, as well as unwanted atmospheric effects.

3.2.1. Goal

The aim of this tutorial is to analyze potential events (earthquake, destruction ...) by highlighting differences between two SAR images of the same portion of the Earth's surface taken at different time using the differential SAR interferometry (DInSAR) technique.

3.2.2. Skills acquired at the end of the training

- Interferometry
- Map deformation
- Building interferograms from two SLC images

3.2.3. Training kit

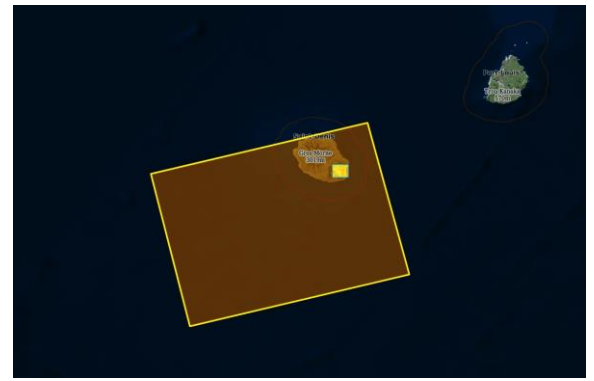
- **Mundi OTB Docker Image**

The docker version made available on Mundi, has been built by Mundi development team and can be found [here](#). (You have to log yourself with your mundi credential to see the content of the mundi shared docker repository). In order to use the OTB docker image on your VM, docker must be installed in all Mundi VM templates and the following commands must be executed replacing `<variable>` according to your needs:

- Log yourself to the docker repository with your mundi website credentials :
`sudo docker login -u <Mundi website user email> -p <Mundi website user password> https://publicreg.mundiwebservices.com/`
- Get the needed docker images on your VM :
`sudo docker pull publicreg.mundiwebservices.com/orfeo_toolbox:<version>`
- Launch your docker in order to use orfeo-toolbox :
`sudo docker run -it -v $(pwd):/data publicreg.mundiwebservices.com/orfeo_toolbox:<version> bash`

- **Dataset**

The data used in this tutorial includes images from Sentinel-1 SLC IW over the eastern side of Réunion island (a French department) in the Indian Ocean at different time. The figure below shows the extent and location of the Sentinel 1 -1 SLC IW data (yellow), and the area of interest over the shield volcano Peak of the Furnace (blue)



Legend

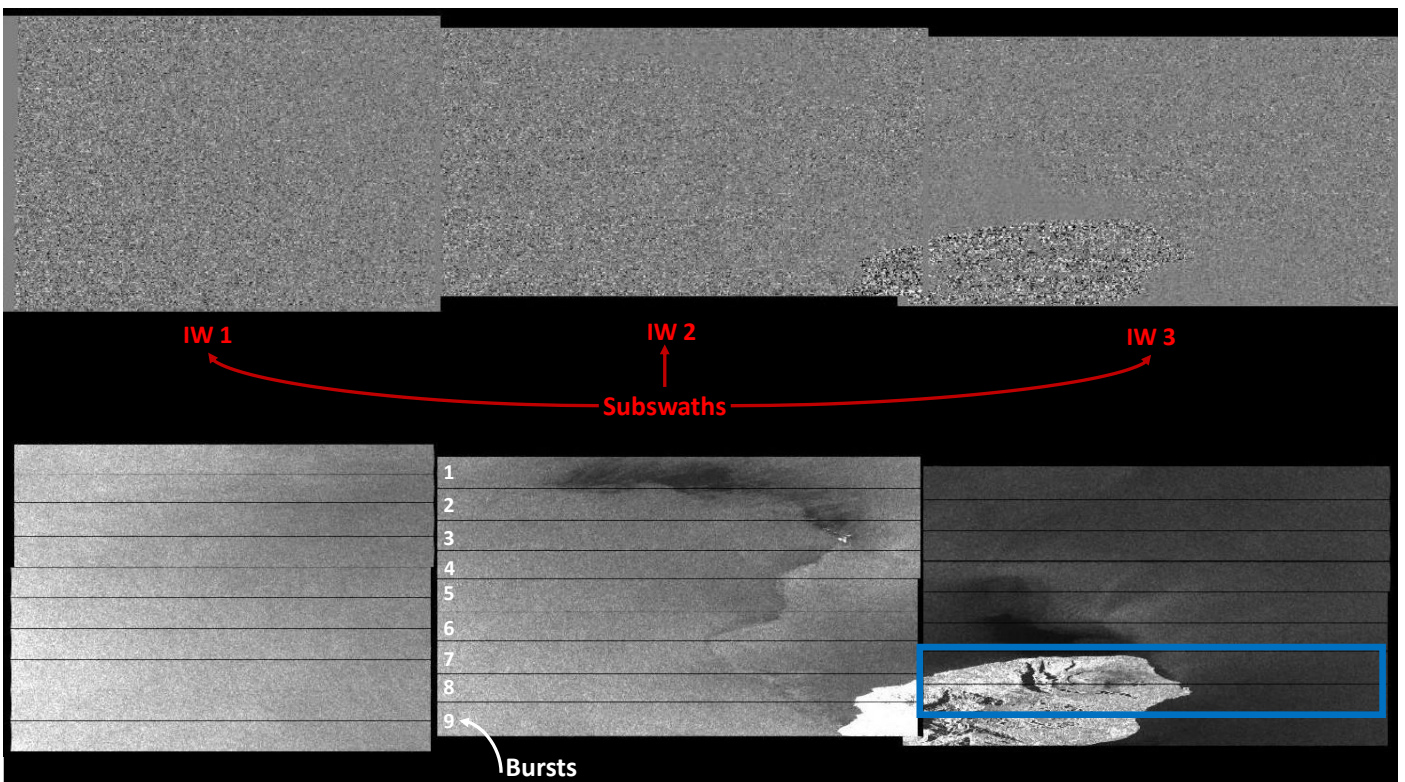
Extent Sentinel-1 SLC data

Area of interest

- Sentinel-1 SLC IW**: Single Look Complex (SLC) Interferometric Wide (IW) Swath products consist of focused SAR data geo-referenced using orbit and attitude data from the satellite and provided in zero-Doppler slant-range geometry. Mundi Web Services provides the complete collection, with fresh free data ONLINE from January 2018 with global coverage, and from January 2017 for Europe. A rolling policy of 12 months for World and 24 months for Europe is currently applied. We are making more data available every day. Discover and view the products with our Geodata UI [here](#)

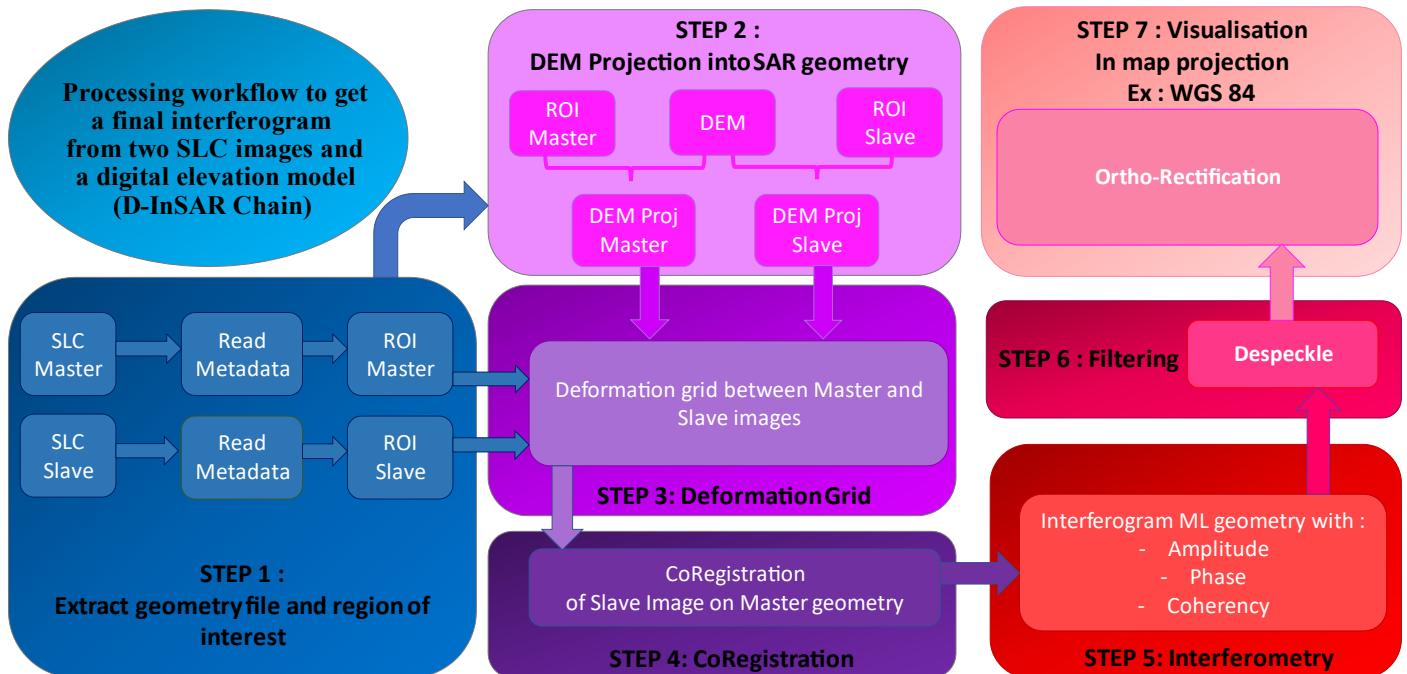
The IW SLC product used in this tutorial contains one image per sub-swath, per polarization channel, for a total of three images. Each sub-swath image consists of a series of bursts, where each burst was processed as a separate SLC image. The individually focused complex burst images are included, in azimuth-time order, into a single sub-swath image, with black-fill demarcation in between.

The figure below shows the Sub-swaths (red) and bursts (white) of S1 IW products as well as the subset used in this tutorial (bleu)



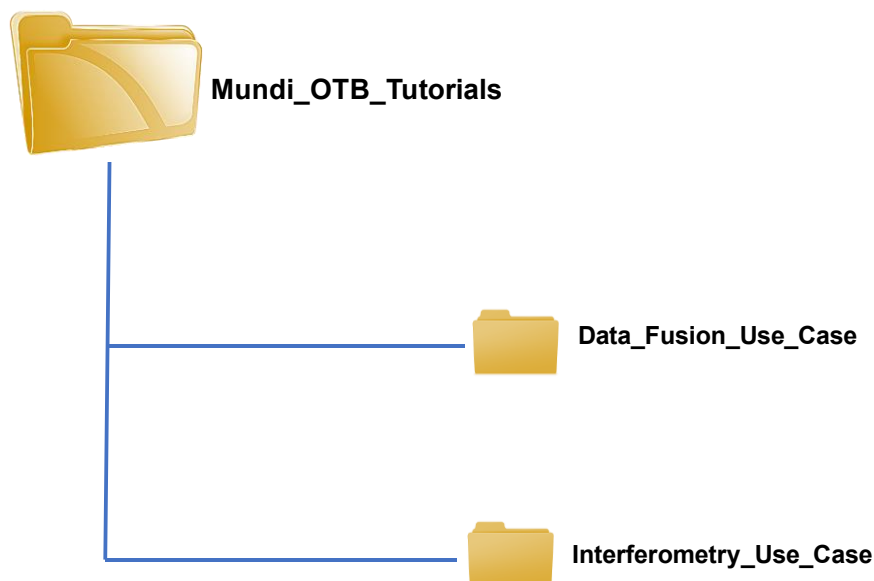
3.2.4. Processing Chains

This section presents the pipeline devoted to the implementation of a legacy processing chain for SAR interferometry called Diapason using OTB applications. This remote module called DiapOTB contains all the necessary steps (depicted in the following diagram) and allows to launch a complete differential SAR interferometry (DInsar) chain using Sentinel-1 data

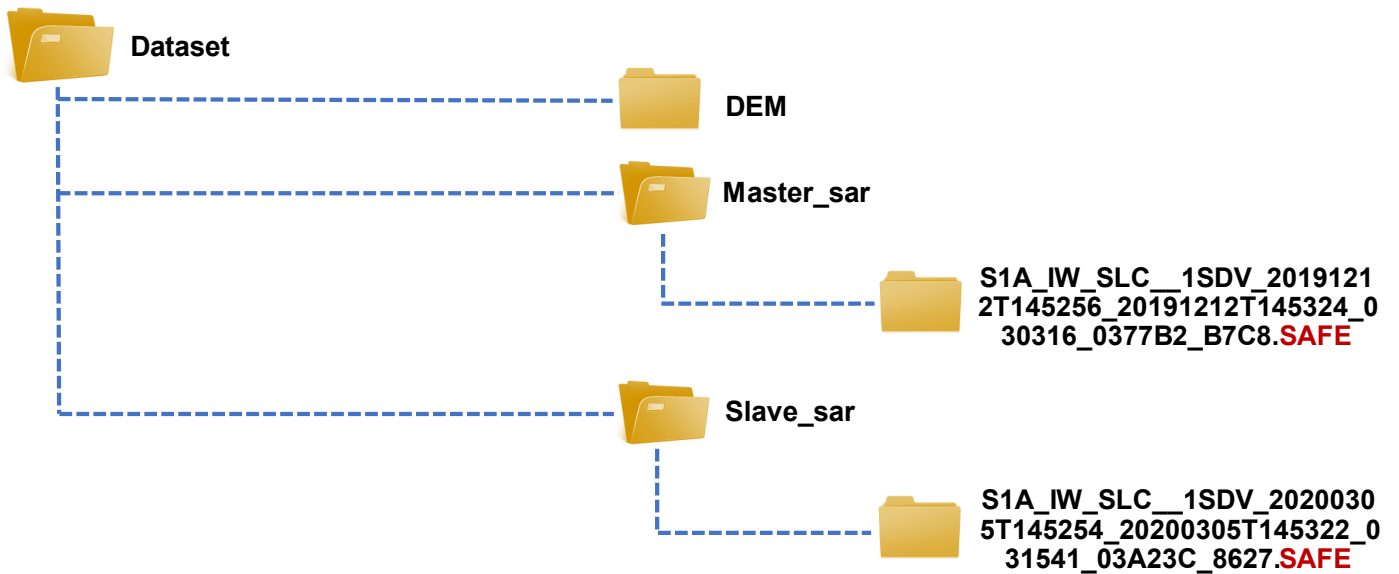


3.2.4.1. Implementation

Mundi Web Services provides you with a script that allows you to run all the process in one step, with the python API. The different files used to run this process can be found in the Mundi tutorial folder, precisely in the Interferometry_use_case folder as follows

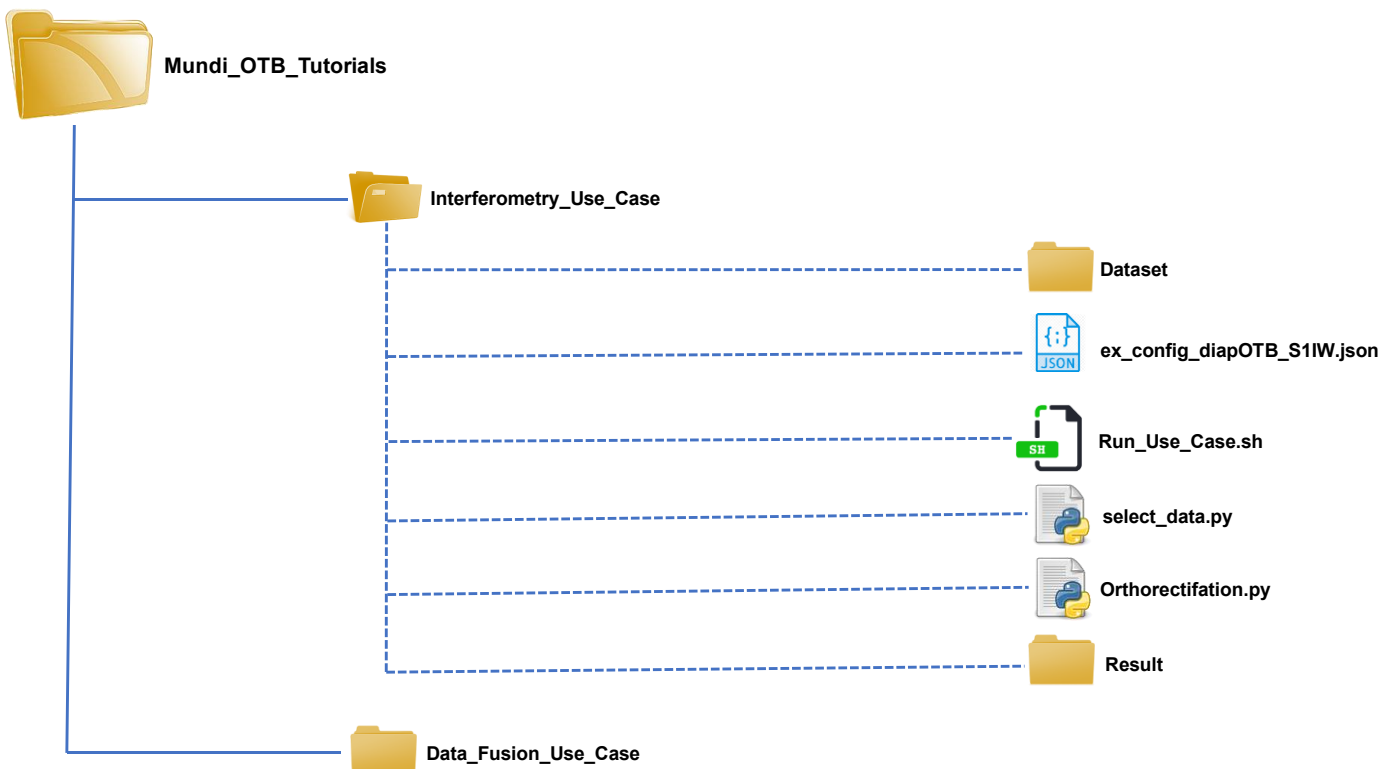


The processing chain needs metadata to launch the applications. Thus, the path to the input images must be into the native directory for each kind of products (i.e SAFE directory for Sentinel-1 products) as follow



To launch this processing chains, only one argument is required as inputs: a configuration file (json format). An interactive script was developed to generate this configuration file following a user's Q&A.

Thus, to execute the chain first, you have to move to the Interferometry use case directory like this:



And then run the following commands

```
# Log yourself to the docker repository with your mundi website credentials :
sudo docker login -u <Mundi website user email> -p <Mundi website user password>
https://publicreg.mundiwebservices.com/

# Get the OTB docker image
sudo docker pull publicreg.mundiwebservices.com/orfeo_toolbox:ubuntu18.04-7.0

# Launch the processing workflow
sudo bash Run_Use_Case.sh
```

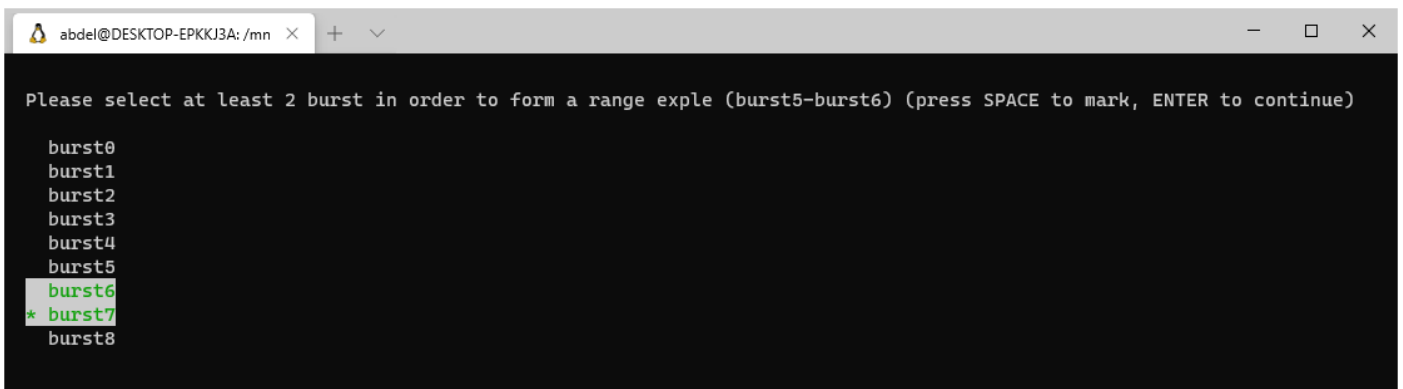
An interactive script will appear to help you set up the configuration file and you will need to answer the following questions as you wish, an example is given here



```
abdel@DESKTOP-EPKKJ3A: /mn x + v - □ x
Please choose the desired subswath:
iw1
iw2
* iw3
```



```
abdel@DESKTOP-EPKKJ3A: /mn x + v - □ x
Please choose the desired polarisation:
* vv
vh
```



```
abdel@DESKTOP-EPKKJ3A: /mn x + v - □ x
Please select at least 2 burst in order to form a range exple (burst5-burst6) (press SPACE to mark, ENTER to continue)
burst0
burst1
burst2
burst3
burst4
burst5
burst6
* burst7
burst8
```



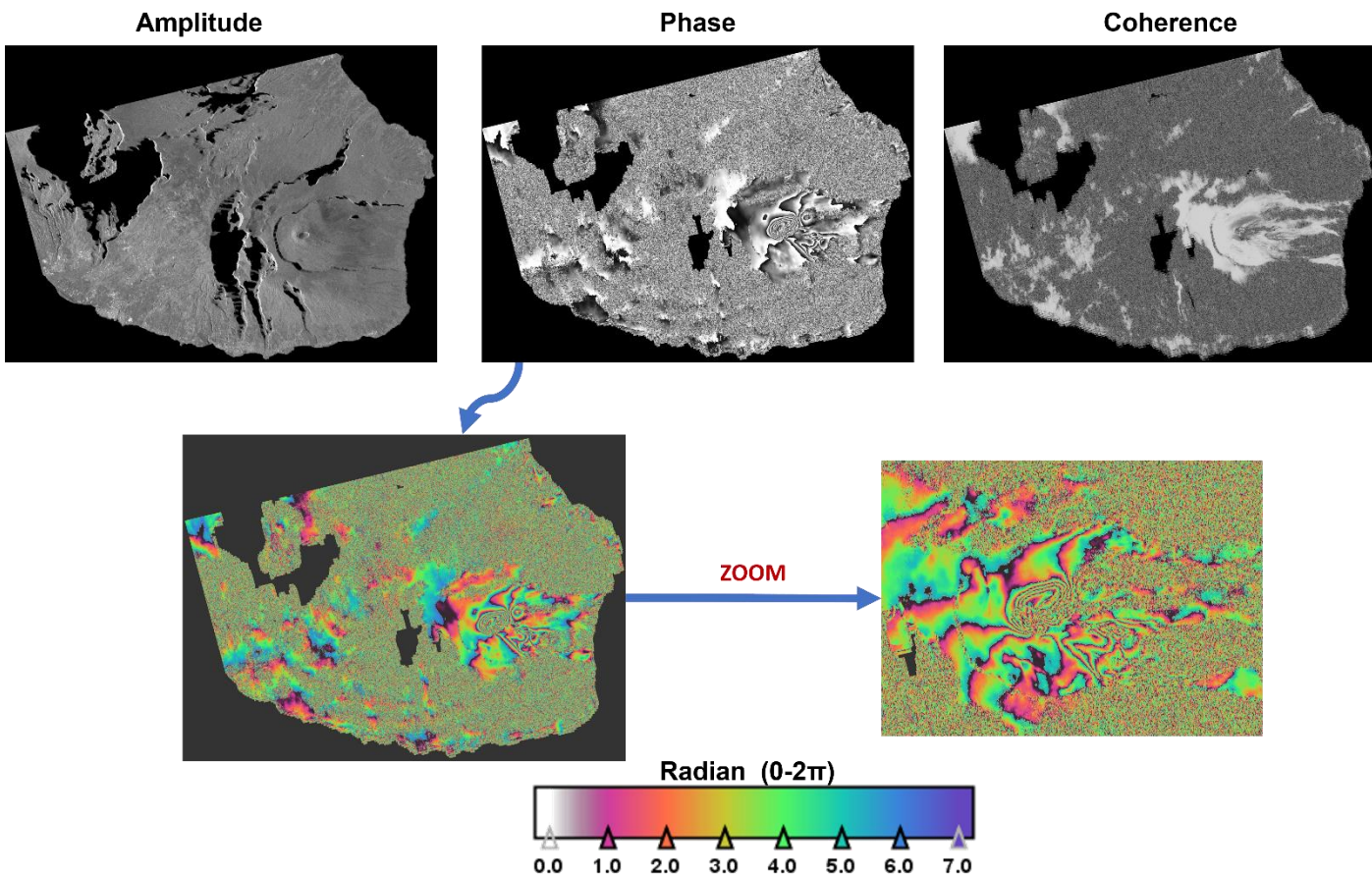
```
abdel@DESKTOP-EPKJ3A: /mn X + v - □ X
2020-08-21 13:50:51 (INFO) DownloadSRTMTiles: Downloading tile S21E055 ...
2020-08-21 13:51:28 (INFO) Mosaic: Default RAM limit for OTB is 256 MB
2020-08-21 13:51:28 (INFO) Mosaic: GDAL maximum cache size is 1281 MB
2020-08-21 13:51:28 (INFO) Mosaic: OTB will use at most 8 threads
2020-08-21 13:51:29 (INFO) Mosaic: Temporary files prefix is: dem/mosaicImage
2020-08-21 13:51:29 (INFO) Mosaic: No feathering
2020-08-21 13:51:29 (INFO): Estimated memory for full processing: 33.9741MB (avail.: 256 MB), optimal image partitioning: 1 blocks
2020-08-21 13:51:29 (INFO): File dem/mosaicImage.tif will be written in 1 blocks of 1201x2401 pixels
Writing dem/mosaicImage.tif...: 100% [*****] (0s)
2020-08-21 13:51:29 (INFO) ManageNoData: Default RAM limit for OTB is 256 MB
2020-08-21 13:51:29 (INFO) ManageNoData: GDAL maximum cache size is 1281 MB
2020-08-21 13:51:29 (INFO) ManageNoData: OTB will use at most 8 threads
2020-08-21 13:51:29 (INFO): Estimated memory for full processing: 32.962MB (avail.: 256 MB), optimal image partitioning: 1 blocks
2020-08-21 13:51:29 (INFO): File Dataset/DEM/dem_mosaicImage_vf.tif will be written in 1 blocks of 1201x2401 pixels
Writing Dataset/DEM/dem_mosaicImage_vf.tif...: 100% [*****] (0s)
data/ex_config_diapOTB_S1IW.json
The output directory does not exist and will be created
Resolution of the input DEM is inferior to 40 meters : A correlation will be used to correct all deformation grids

Beginning of DiapOTB processing (S1 IW mode)

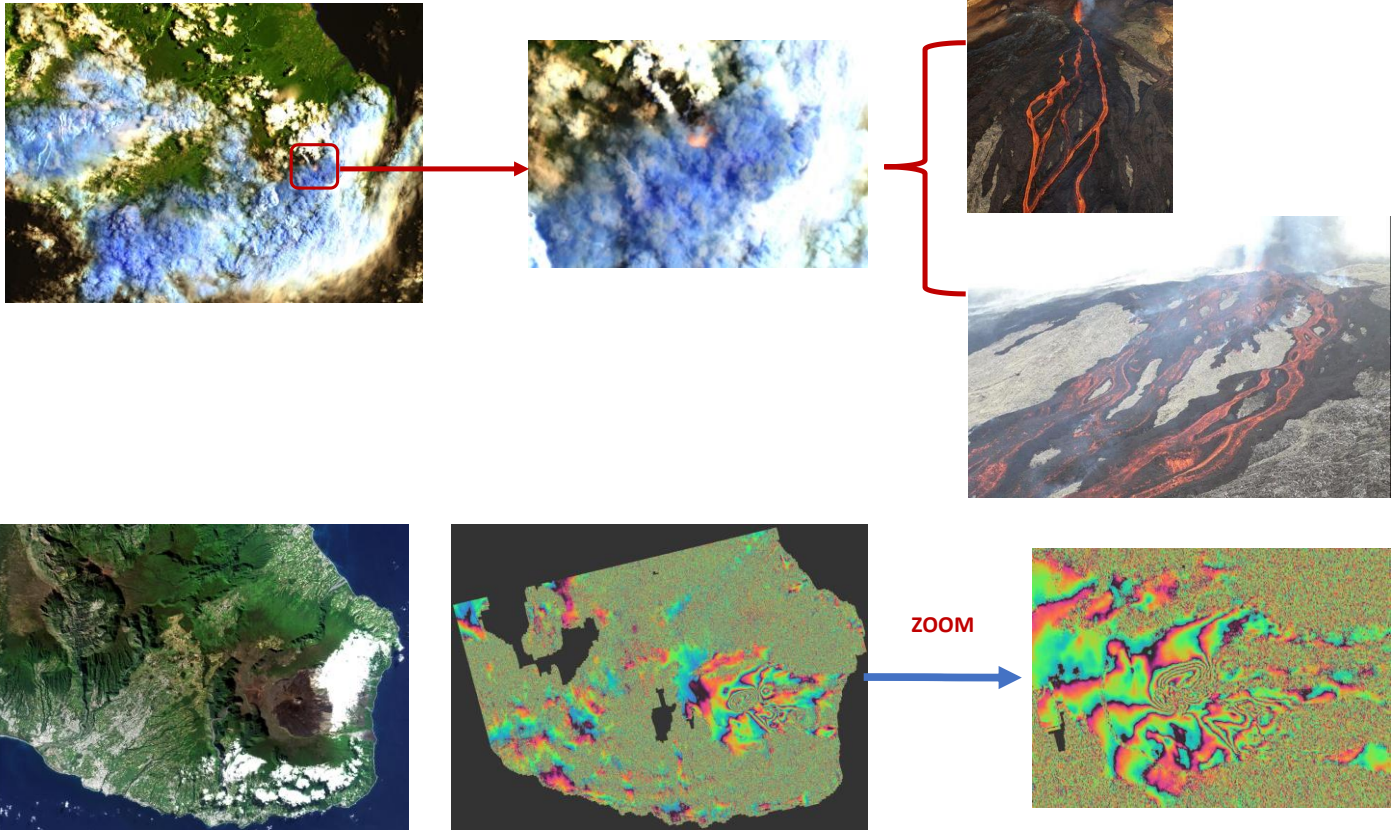
Master Pre_Processing chain

Slave Pre_Processing chain
```

3.2.4.2. Results



volcanic eruption 10/02/2020



These illustrations show terrain displacements after the volcanic eruption of Peak of the Furnace (eastern side of Réunion island) that occurred on February 10th 2020, computed on MUNDI VM with the remote module DiapOTB from a set of ascending Sentinel-1 pair (resp 2019/12/12 and 2020/03/05). One fringe represents a displacement of half the wavelength in Line Of Sight (LOS), that is about 2.8 cm for Sentinel-1.