OGRS2012 : Exploit Pleiades PHR data with the ORFEO ToolBox

Manuel Grizonnet (CNES), Julien Michel (CNES)

October 23, 2012

Contents

1 Foreword

1.1 About the data

The images used during these exercise are extracts from Pleiades demonstration products, made available for evaluation purpose. To get the full products please refer to this website. Products used are:

- `Pleiades Pan-sharpened ORTHO Compression REGULAR`
- `Pleiades TRISTEREO Bundle PRIMARY`

They are covered by a cnes copyright.
Other data needed for some exercises, as well as solution scripts can be found in the data package.

1.2 About the software

To perform the exercises, you will need to have the following software installed:

- **Orfeo ToolBox** 3.14 or later, including applications
- **Monteverdi** 1.6 or later

- **QGis** 1.8 or later

  For **Orfeo ToolBox** and **Monteverdi** installation, you can refer to the installation from the Orfeo ToolBox Cookbook.

  For **QGis** installation, please refer to **QGis** documentation, which can be found on the project website.

2 Exercises

2.1 Monteverdi and OTB applications

2.1.1 Description

Abstract This exercise will get you familiar with the use of **Monteverdi** and **OTB applications**.

Pre-requisites

- Basic knowledge of remote sensing and image processing,
- Basic knowledge of command-line invocation.

Achievements

- Visualize data in **Monteverdi**,
- Basic processing in **Monteverdi**,
- Basic processing with **OTB applications** in graphical mode,
- Basic processing with **OTB applications** in command-line mode.

2.1.2 Steps

Monteverdi: data opening and visualisation In this part of the exercise, you will use the following data: `phr_pxs_melbourne.tif`

1. Run **Monteverdi**: open a terminal and run the following command:

   ```
   $ monteverdi
   ```

2. Open the image (use the *File/Open dataset* menu)
3. Find how to display the image (there are two ways of doing it).
4. Navigate into the image:
   (a) Change the full resolution displayed area
   (b) Change the zoom displayed area,
   (c) Change the zoom level,
   (d) What are the information displayed about the current pixel under mouse pointer ?

5. Using the viewer control panel, in the *Setup* tab:
   (a) Change set-up to visualize the 4th band,
   (b) Change set-up to visualize in false color mode (screen red: near infra-red channel, screen green: red channel, screen blue: green channel).
   (c) Change set-up to come back to natural colors
   (d) Enhance contrast with respect to the full area,
   (e) Enhance contrast with respect to the zoom area,
   (f) Come back to default contrast enhancement.

Tips and Recommandations:

- You can use keyboard arrows to navigate into images as well,
- Pleiades bands order is red channel, green channel, blue channel, near infra-red channel.

Monteverdi: basic processing In this part of the exercise, you will use the following data:
`phr_xs_melbourne.tif`

1. Open the image in **Monteverdi**.
2. Find the *BandMath* module in the menu. Open the image in this module. What kind of processing is offered ?
3. Using this module, compute the NDVI of the image:

$$NDVI = \frac{NIR - RED}{NIR + RED} \tag{1}$$

Visualize the input image and the mask in the same viewer.

1. Using this module, build a mask of pixels whose Digital Number (DN) in the NIR channel is lower than 150. Visualize the input image and the mask in the same viewer.
2. Using this module, build a mask of pixels whose DN is upper than 1000 in all spectral bands. Visualize the input image and the mask in the same viewer.
3. Using the *Concatenate* module, build a composite RGB image with the mask of high values in the red channel, the mask of low NIR values in the blue channel and the NDVI in the green channel.
4. Using the *Color Mapping* Module, build a composite RGB image of the NDVI that allows for better image interpretation.

Tips and Recommandations:

- NDVI values are within -1 and 1, but the range can be much more narrow.

OTB applications: Graphical and command-line mode

1. Run the following command:

```
$ otbcli_OrthoRectification
```

   And then

```
$ otbgui_OrthoRectification
```

   What do you observe ?

2. How many **OTB applications** are currently available ?
3. How can you get help and documentation about applications ?

OTB applications: Basic processing

1. Use the **OTB applications** to produce the same results as steps 3 to 7 as with **Monteverdi** in section Monteverdi: basic processing.

Homework

1. How can we load or visualize images directly from command-line using **Monteverdi** ?
2. Is there another way to compute radiometric indices like NDVI with the **OTB Applications** ?
3. Learn about the *Python* access to **OTB Applications** and write a python script performing the same steps as in section OTB applications: Basic processing.

2.2 Segmentation

2.2.1 Description

Abstract This exercise will get you familiar with the OTB **Segmentation** application. You will learn how to produce a raster segmentation output with different algorithms and how to scale up to larger input images by producing vector outputs.

Pre-requisites

- Basic knowledge on OTB applications and QGis usage
- Basic knowledge on image segmentation
- Basic knowledge on GIS vector file formats

Achievements

- Usage of the OTB **Segmentation** application,
- Segmentation of large raster and import the results in a GIS software.

2.2.2 Steps

Getting familiar with the Segmentation application

1. Run the command-line and graphical version of the application
2. Read the documentation. What are the three segmentation methods available ?
3. What are the two output modes ?

Simple segmentation in raster mode In this part of the exercise, you will use the following data: `segmentation_small_xt_phr.tif`

1. Run the **Segmentation** application in *raster* mode, using the connected components filter and a thresholding condition on the spectral distance
2. View the resulting segmentation in **Monteverdi**. What do you see ?
3. Use the **ColorMapping** application to enhance the rendering of the result:
   (a) Try the *optimal* method
   (b) Try the *image* method
4. Try different connected components conditions and see how they influence the results. You can try to change the distance threshold for instance, or look into the documentation for other keywords.

Tips and Recommandations:

- Use the **distance** keyword in the expression to denote spectral distance
- Pay attention to the output image type

More segmentation algorithms In this part of the exercise, you will use the following data: `segmentation_small_xt_phr.tif`

1. Run the **Segmentation** application in *raster* mode again, but this time use the Mean-Shift filter. Use the **ColorMapping** application to visualize the results.
   (a) Try the default parameters first
   (b) Try to change the parameters and see how it influences the results. The most important parameters are the spatial and the range radius.
2. Run the **Segmentation** application in *raster* mode again, but this time use the Watershed filter. Use the **ColorMapping** application to visualize the results.
   (a) Try the default parameters first
   (b) Try to change the parameters and see how it influences the results.
3. Compare the best results from the three algorithms. Keep the best segmentation result you had for Exercise 3.

Tips and Recommendations:

- There are two implementations of the Mean-Shift filter. Edison is the original implementation from the Mean-Shift paper authors.

Going big: the vector mode In this part of the exercise, you will use the following data:
`segmentation_large_xt_phr.tif`

1. Run the **Segmentation** application in *raster* mode again, using the best parameters you had in previous section, on the large image. Look at computer resources. What happens ?
2. Run the **Segmentation** application again, this time in *vector* mode, and **disable the stitching option**. Look at computer resources. What happens ?
3. Open the result of the input image and the segmentation file in **QGis**. Tune **QGis** to allow for proper visualization (see Tips and Recommendation). What do you see ?
4. Run the **Segmentation** application again, this time in *vector* mode, and **enable the stitching mode**. Write the results to a different file and load it into the **QGis** project as well. What is the effect of the **stitch** option ?

Tips and Recommendations:

- Computer resources can be monitored by running `top` in another terminal
- Hit `Ctrl C` to interrupt the processing
- Use the *sqlite* file format to store vector outputs (`.sqlite` file extension)
- In **QGis**, one can import both raster and vector layers
- In **QGis**, one can tune raster layers rendering the following way:
  - Right-click on the layer, select *Properties*
  - Go to the *style* tab
  - Select *Use standard deviation*
  - In *Contrast enhancement*, select *Stretch to MinMax*
- In **QGis**, one can tune vector layers rendering the following way:
  - Right-click on the layer, select *Properties*
  - In the *style* tab, select *Change*
  - As *Symbol layer type*, select *Outline: Simple line*
  - You might change the color as well
- In **QGis**, you can save your project to a file and avoid having to reset those parameters

Homework

1. In *vector* mode, study the effect of the *tilesize*,/simplify/ and *minsize* option.
2. Using the **Segmentation** application (and maybe other OTB applications), how can we segment everything but vegetation ?
3. Using the **Segmentation** application (and maybe other OTB applications), how can we deal with segmentation of high reflectance structures ?

2.3 Learning and classification from pixels

2.3.1 Description

Abstract This exercise will get you familiar with the OTB pixel based classification applications. You will learn how to train a SVM classification model from Pleiades images and a set of training regions. You will then learn how to apply this model to images and produce shiny classification maps.

Pre-requisites

- Basic knowledge on OTB applications and QGis usage
- Basic knowledge on image supervised classification
- Basic knowledge on GIS vector file formats

Achievements

- Usage of the OTB Classification applications
- Classification of large images
- Import of results in a GIS software

2.3.2 Steps

In this part of the exercise, you will use the following data:
`melbourne_ms_toa_ortho_extract_small.tif`

Produce and analyze learning samples

- Use Qgis to produce polygons for 5 classes (vegetation, roads, soil, buildings and water)
- Export this vector layer in shapefile
- What is the label corresponding to the class **water** in the shapefile? An example set of learning samples is provided for the exercise in *training.shp*

Tips and Recommendations:

- Note the field name of the shapefile which contains the label. You will need to provide this field in the training application

Estimate image statistics In order to make these features comparable between each images, the first step is to estimate the input images statistics. These statistics will be used to center and reduce the intensities (mean of 0 and standard deviation of 1) of training samples from the vector data produced by the user.

- Use the **ComputeImagesStatistics** to compute statistics on the image
- What is the mean of the red band?
- The extract provided has been converted from DN to milli-reflectance. For what reasons, is it advised to do so when performing multiple images classification?

Estimate classification model using the Support Vector Machine algorithm The **TrainSVMImagesClassifier** application performs SVM classifier training from multiple pairs of input images and training vector data. Samples are composed of pixel values in each band optionally centered and reduced using XML statistics file produced by the **ComputeImagesStatistics** application. We will use this application with only one image in this exercise.

- Use the **TrainSVMImagesClassifier** to produce SVM model
- Which kernel is used by default in the application?
- What is the measured accuracy?

Apply classification model

- Use the **ImageSVMClassifier** to apply the classification model to the input image
- What is the output of the application?
- Bonus : Use the same model to apply the classification to the other extract

  `melbourne_ms_toa_ortho_extract_large.tif`

Produce printable classification map We are now going to produce a printable classification map using the **ColorMapping** application This tool will replace each label with an 8-bits RGB color specified in a mapping file. The mapping file should look like this :

```
$ # Lines beginning with a # are ignored
1 255 0 0
```

- Produce your custom look-up table (LUT)
- Use this LUT to produce a printable classification map (in PNG format)
- Overlay this map on the input image in QGIS. Comment on the classification results.

Homework

- Produce classification model with different kind of SVM kernels. Comment different accuracies obtained?
- Going big: Apply this classification on the pan-sharpened image over Melbourne

2.4 Learning and classification from objects

2.4.1 Description

Abstract This workshop will introduce you to the **Object Labeling** module of **Monteverdi**. You will learn how to use the module and see the influence of different features on classification results. You will also experiment with a simple active learning implementation on objects.

Pre-requisites

- Basic knowledge of Object Based Image Analysis
- Basic knowledge on learning and classification

Achievements Being able to use the **Object Labeling** module of **Monteverdi**.

2.4.2 Steps

The preliminary segmentation In this part of the exercise, we will use the following data:
`phr_pxs_melbourne_xt_small.tif`
`phr_pxs_melbourne_xt_small_segmentation.tif`

1. Use the **ColorMapping** application to enhance the visualization of the segmented image (you can use the *optimal* and *image* modes as learned in the segmentation exercise).
2. Analyze the color-mapped segmentation results. For which kind of objects is the object based classification likely to work well ? For which kind of objects is it likely to perform badly?

Object Labeling module - basics

1. Open both the image and the segmentation image in **Monteverdi**.
2. Open the **Object Labeling** module from the *learning* menu, and load the image and the segmentation inside the module.
3. What is the purpose of each tab on the left side of the module?
4. In the *Objects* tab, create a new class. You can change its color and its name.
5. Right-click on an object of interest in the image. What happens?
6. Right-click a second time inside the selected object. What happens?
7. Add a few more objects to the current class.
8. Create a new class and add some objects to it.
9. Go to the *Features* tab, uncheck all features but the mean radiometric values.
10. Go to the *Learning* tab and click on classify. What happens?
11. Click on the *Save/Quit* button. What kind of outputs is produced by the module?

Tips and Recommendations:

- Choose two simple classes for this part of the exercise (for instance a *Water* class and a *Land* class)
- Use the navigation map to change the displayed area
- You can change the opacity of the classification layer as well as of the selected objects layer so as to better analyze the results.
- You can also clear the classification layer.

Object Labeling module - advanced In this part of the exercise, we will use these additional files: `samples.xml` and `parameters.xml`

1. Load again the image and the segmentation inside the module.
2. Load the samples file using *File/Load Samples*. What are the different object classes loaded ? How many samples per classes are used ?
3. Uncheck all features except from radiometric means:
   - Band1::Mean
   - Band2::Mean
   - Band3::Mean
   - Band4::Mean
4. Perform the classification. What are the objects in the image that are badly classified because of missing classes ?
5. What are the objects in the image that are poorly classified because they are badly segmented or too complex ?
6. Try to enhance the classification by adding missing classes.
7. Try to enhance the classification by adding new features.

Tips and Recommendations:

- The **Object Labeling** module is quite memory consuming. Depending on the available memory on your system, you might want to restart **Monteverdi**.

Object Labeling module - active learning

1. In the *Objects* tab, click on the *Sample* button in the lower-left area. This will show you difficult samples by using the *margin sampling* technique.
2. What kind of segments are considered by the algorithm as hard to classify ?
3. Try to create a *Trash* class to handle noise segments.
4. Perform a few more iteration of active learning. What do you observe ?

2.5 Elevation map from stereo pair

2.5.1 Description

Abstract This exercise will guide get you familiar with the set of OTB applications which allow to compute elevation map from a stereo pair of optical images. You will learn how to :

- re-sample for stereo pair in epipolar geometry to reduce the stereo correspondences to a 1D problem
- Perform block matching between the 2 images to extract the disparity (related to the elevation)
- Filter disparities using correlation metric analysis

Pre-requisites

- Basic knowledge on OTB applications
- Basic knowledge on epipolar geometry. Epipolar geometry is the geometry of stereo vision (see here). The operation of stereo rectification determines transformations to apply to each image such that pairs of conjugate epipolar lines become collinear, parallel to one of the image axes and aligned. In this geometry, the objects on a given row of the left image are also located on the same line in the right image.

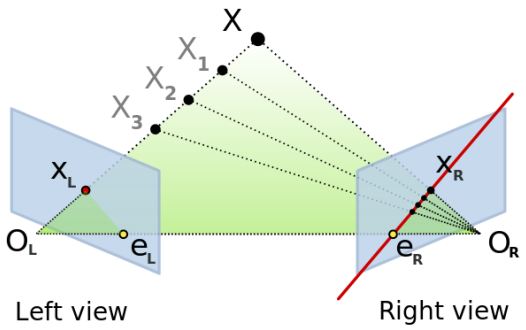


Figure 1: Epipolar geometry

- Basic knowledge of stereoscopic reconstruction

Achievements

- Usage of stereoscopic reconstruction applications
- Stereo reconstruction based on Pleiades stereo images pair

2.5.2 Steps

From images to epipolar geometry In this part of the exercise, you will use the following data: `tristereo_melbourne_1_small.tif and tristereo_melbourne_2_small.tif`

1. Run the command-line and graphical version of the **StereoRectificationGridGenerator** application
2. What are the two outputs of the applications?
3. Use the application to generate two re-sampling grids. Which OTB application allows to re-sample the two input images using these grids?
4. Use this application to resample input stereo pair in epipolar geometry, open the 2 re-sampled images. What do you see ?

Tips and Recommandations:

- Perform the grids estimation using and average elevation of 20.45m (**epi.elevation.avg.value** keyword)
- Stereo-rectification deformation grid only varies slowly. Therefore, it is recommended to use a coarser grid (higher step value) in case of large images (**epi.step** keyword)
- Note the size of the images in epipolar geometry (output by the application)

Improvement of epipolar geometry Pleiades data can be orthorectified to absolute accuracies of about 10 meters, as a consequence there is still a need to improve the geometric accuracy (this is the case for all satellite imagery). For orthorectification purpose, it is achieved by optimizing sensor modelling with Ground Control Points. An other way to do this in case of superimposition of multiple images is to produce homologous points on each images and refine with these points the co-localisation function. It allows to improve the geometric accuracy and to produce consistent epipolar images.

We provide for the next questions a refined version of the stereo pair:

`tristereo_melbourne_1_small_ref.tif`
`tristereo_melbourne_2_small_ref.tif`

1. Recompute epipolar geometry with the new stereo pair (post-fixed by *_ref.tif*). Open the 2 versions of epipolar couples (total of 4 images). What differences do you notice between the two images pair?
2. Combined the 2 images to create a 3D anaglyph (left image on the red channel and the right image on the green and blue channel). Visualize the anaglyph with anaglyph glasses.

We will use this images in the next questions.

Block matching We are going to perform stereo pair block matching on the two images using the **BlockMatching** application.

1. Run the command-line and graphical version of the **BlockMatching** application. What are the mandatory parameters?
2. Propose manual or automatic methodologies to estimate the interval of disparities in vertical or horizontal direction.
3. Use these parameters to generate a disparity map and open the result with Monteverdi. What do you notice?

   Tips and Recommendations:

   - Discard pixels with no-data (0 in our case) value using the parameter **-mask.nodata**

Advanced Block matching : refine disparity map We are going to try now to improve the quality of the disparity map using options available in the **BlockMatching**.

1. Use the Normalized Cross Correlation and output the metric value using the io.outmetric option. Open the metric image, which values of correlation corresponds to a good disparity value ?
2. Use the option mask.variancet to discard pixels whose local variance is too small (the size of the neighborhood is given by the radius parameter)
3. Use the **BandMath** application to only keep horizontal disparity with high correlation value.

From disparity map to ground elevation Use the **DisparityMapToElevationMap** to transform the disparity map into an elevation map.

1. At which height approximately do cricket players play in the stadium?
2. What is approximately the height of the stadium?

Tips and Recommandations:

- Reuse the same average elevation of 20.45m
- Bonus : produce a mask using the **BandMath** application to discard pixels with low correlation values using the parameter **io.mask**

Homework

1. Try refinement steps to improve epipolar geometries (available soon in OTB -> 3.16 version)
2. Perform disparity coherence analysis by comparing disparity maps obtained by switching the left and right images
3. Re-compute disparity maps with sub-pixel precision block-matching
4. Use median filter to get a smoother disparity map

3 Solutions

3.1 Monteverdi and OTB-Applications

3.1.1 Monteverdi: data opening and visualisation

Item 3 To load an image into **Monteverdi** viewer module, you can either:

- Right-click on the image and select *Display in viewer*,
- In the menu bar, select *Visualization/Viewer*, select the corresponding image and push *Ok*.

The latter allows to load multiple images into a single viewer.

Item 4 The lower left text area displays information on the image and on the pixel under the mouse pointer:

- The current position in image,
- The image size,
- The channel displayed,
- The pixel values,
- The estimated ground spacing,
- The geographic position (if available),
- The current location (if available).

3.1.2 Monteverdi: basic processing

Item 2 The **BandMath** module allows to do advanced band calculations using the syntax from muParser .

Item 3 To compute the NDVI, use the following **BandMath** expression:

```
(im1b4-im1b1)/(im1b4+im1b1)
```

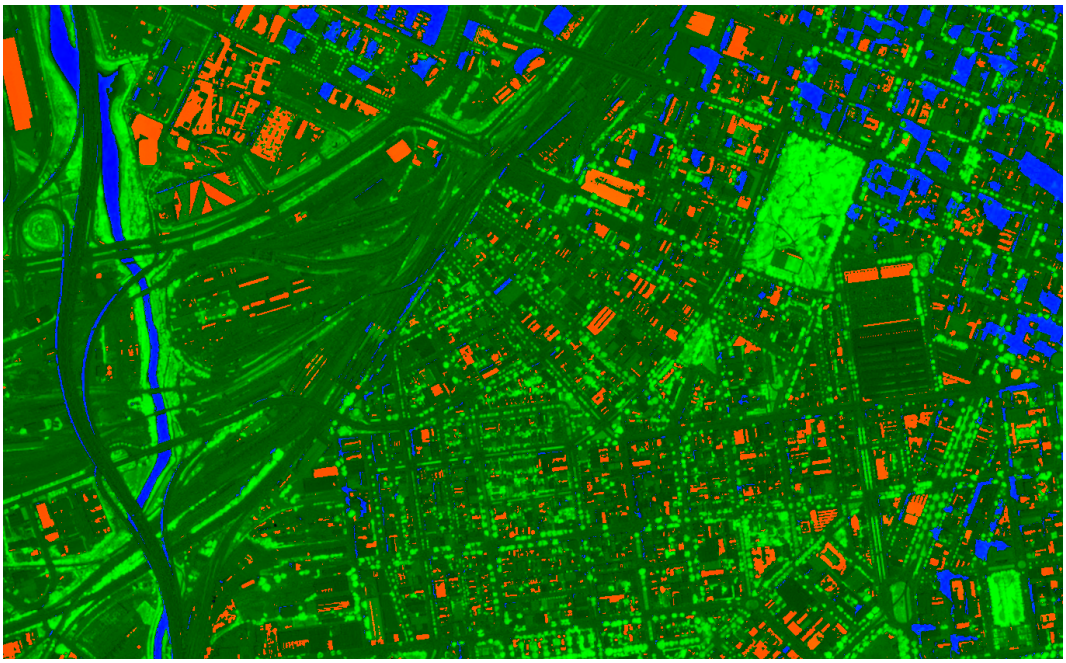
Item 4 To build a mask of pixels whose DN in the NIR channel is lower than 150, use the following **BandMath** expression:

```
if(im1b4<150,255,0)
```

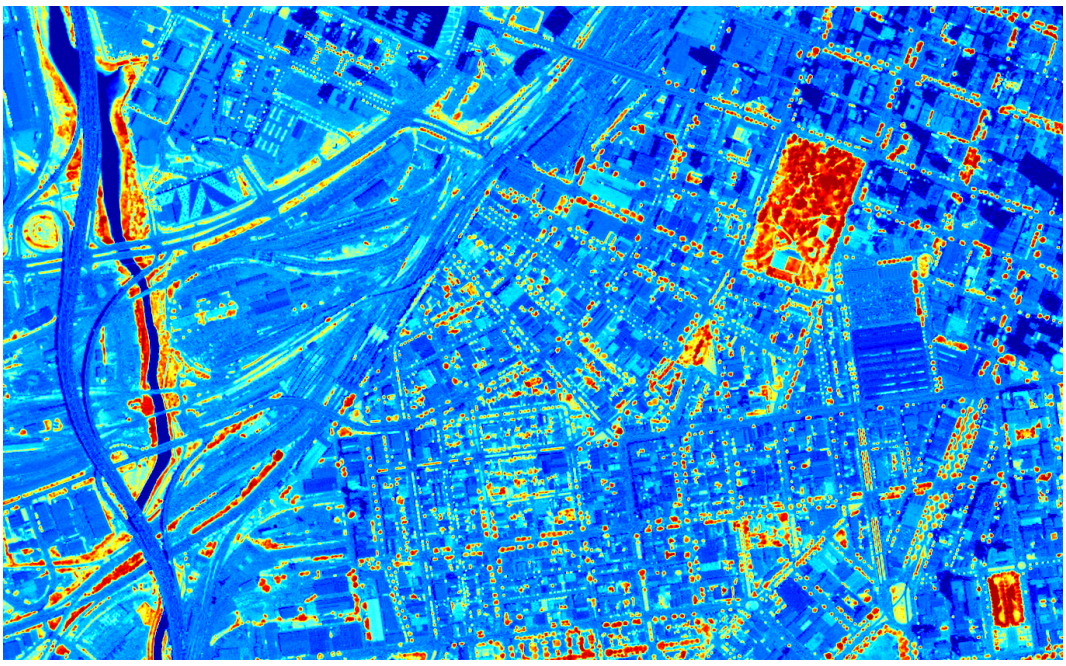
Item 5 To build a mask of pixels whose DN is upper than 1000 in all spectral bands, use the following **BandMath** expression:

```
if(min(im1b1,im1b2,im1b3,im1b4)>1000,255,0)
```

Item 6 In the menu bar, select *File/Concatenate images*, and loads the three **BandMath** module outputs. The resulting image can be displayed in the viewer and will look like this:



Item 7 In the menu bar, select *Visualisation/Color Mapping* and load the NDVI output from the **BandMath** module. Set a mapping range from -0.2 to 0.7 so as to adapt to NDVI range, and select the *Jet* color map. The resulting image can be displayed in the viewer and will look like this:



3.1.3 OTB applications: Graphical and command-line mode

Item 1 The first command runs the command-line version of the **Orthorectification** application, the second one runs the graphical version.

Item 2 There are 59 applications available in OTB 3.14.1.

Item 3 There are several ways to get help and documentation:

- Running the command-line version of the application displays a short description of the parameters, and also gives a link to the documentation on the OTB website,
- Running the graphical version of the application shows a *Documentation* tab where extensive documentation of parameters can be found.
- Last, the complete applications documentation can be found in the Orfeo ToolBox Cookbook.

3.1.4 OTB applications: Basic processing

Item 1 Here is the set of commands to reproduce the processing from section Monteverdi: basic processing.

First, we compute the NDVI with the **BandMath** application:

```
$ otbcli_BandMath -il phr_xs_melbourne.tif
  -out ndvi.tif float -exp "(im1b4-im1b1)/(im1b4+im1b1)"
```

Then, we compute the mask of pixels whose DN in the NIR channel is lower than 150:

```
$ otbcli_BandMath -il phr_xs_melbourne.tif
  -out lownir.tif uint8 -exp "if(im1b4<150,255,0)"
```

Next, we compute the mask of pixels whose DN is upper than 1000 in all spectral bands:

```
$ otbcli_BandMath -il phr_xs_melbourne.tif
  -out high.tif uint8
  -exp "if(min(im1b1,im1b2,im1b3,im1b4)>1000,255,0)"
```

Please note that for masks using a *uint8* data type is enough, while for NDVI a floating point data type is needed.

Now, we can concatenate all outputs in a single map with the **ConcatenateImages** application:

```
$ otbcli_ConcatenateImages -il high.tif ndvi.tif lownir.tif
  -out map1.tif float
```

Finally, we can create a color-mapping of the NDVI using the **ColorMapping** application:

```
$ otbcli_ColorMapping -in ndvi.tif -out map2.png uint8
  -method continuous -method.continuous.min -0.2
  -method.continuous.max 0.7 -method.continuous.lut jet
```

3.1.5 Homework

Item 1 From the command-line, running

```
$ monteverdi -in  phr_xs_melbourne.tif
```

will open the image in **Monteverdi** and display it in the viewer, and

```
$ monteverdi -il  phr_xs_melbourne.tif ndvi.tif
```

allows to open a list of images in **Monteverdi**.

Item 2 In **OTB Applications**, there is a **RadiometricVegetationIndices** application that allows to compute several indices including the NDVI.

Item 3 Please refer to this chapter of the **Cookbook** to learn more about the *Python* interface.

3.2 Segmentation

3.2.1 Getting familiar with the Segmentation application

Item 1 To get the command-line help, run

```
$ otbcli_Segmentation
```

To Get the graphical version of the **Segmentation** application, run

```
$ otbgui_Segmentation
```

Item 2 There are three segmentation methods available in the application:

- Mean-Shift (two different implementations)
- Watershed (ITK implementation)
- Connected-Components

Item 3 There are two outputs available in the application:

- The raster mode allows to segment a small image and produces a raster where each component of the segmentation is labeled with a unique integer,
- The vector mode allows to segment larger images and produces a vector file where each segment of the segmentation is represented by a polygon.

3.2.2 Simple segmentation in raster mode

Item 1 Here is the command-line to run, using a threshold of 30 on the spectral distance:

```
$ otbcli_Segmentation -in segmentation_small_xt_phr.tif
  -filter cc -filter.cc.expr "distance < 30"
  -mode raster -mode.raster.out first_cc.tif uint32
```

Please note that we use `uint32` as the output type so as to be sure to have enough unique labels for the whole segmentation.

Item 2 The segmentation result is difficult to visualize because neighboring segments are likely to be labeled with very close labels. One can notice the brightness gradient from top to bottom corresponding to globally increasing labels.

Item 3 The following command-line allow to use the **ColorMapping** application in optimal mode:

```
$ otbcli_ColorMapping -in first_cc.tif
  -out first_cc_color_optimal.png uint8
  -method optimal
```

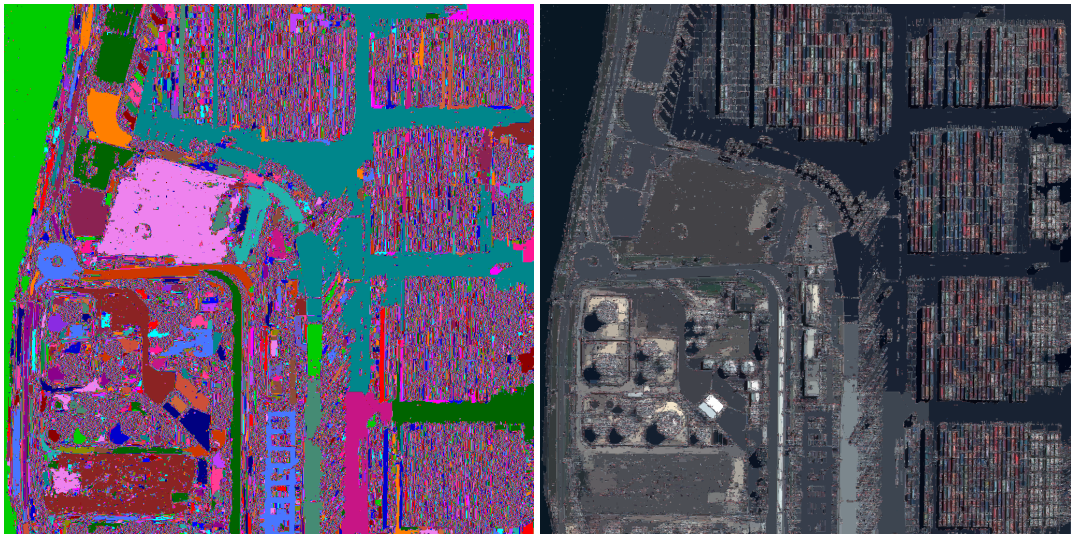
The *optimal* color-mapping method allows to colorize each segment with a color maximizing contrast with the color of its neighbors. Please note that we use `uint8` as the output type because the **ColorMapping** application produces 8-bits data that can be directly viewed by any image viewer.

Looking at the colorized image with the *optimal* look-up table, we can now see that the result is over-segmented.

```
$ otbcli_ColorMapping -in first_cc.tif
  -out first_cc_color_image.png uint8
  -method image -method.image.in segmentation_small_xt_phr.tif
```

The *image* color-mapping method allows to colorize each segment with its mean color in the original image, which gives a more realistic rendering. Note that since the results are over-segmented, the application will output a huge amount of text to the terminal.

Here are the results of the *optimal* (left) and *image* (right) methods:

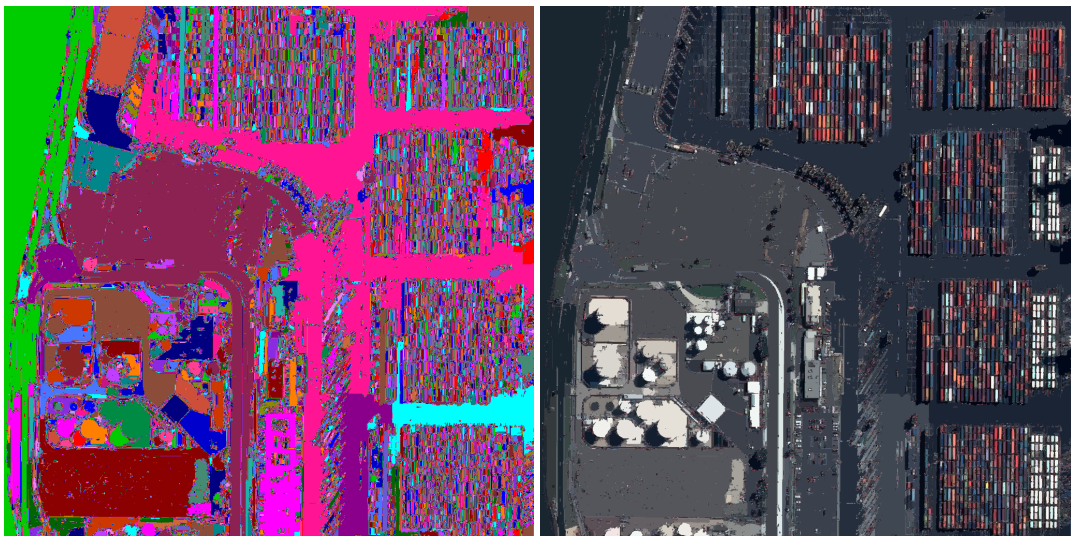


Item 4 Here is another example: the following command-line will segment together pixels that either:

- Have a spectral distance lower than 30,
- Have both an intensity value greater than 400 and a spectral distance lower than 50,
- Have both an intensity value greater than 1000,
- Have both a near infra-red value lower than 150.

```
$ otbcli_Segmentation -in segmentation_small_xt_phr.tif
  -filter cc -filter.cc.expr "distance<30
   or (intensity_p1>400 and intensity_p2 > 400 and distance<50)
   or(intensity_p1 >1000 and intensity_p2>1000
   or (p1b4 <150 and p2b4<150))"
  -mode raster -mode.raster.out second_cc.tif uint32
```

Here are the color-mapping results:



3.2.3 More segmentation algorithms

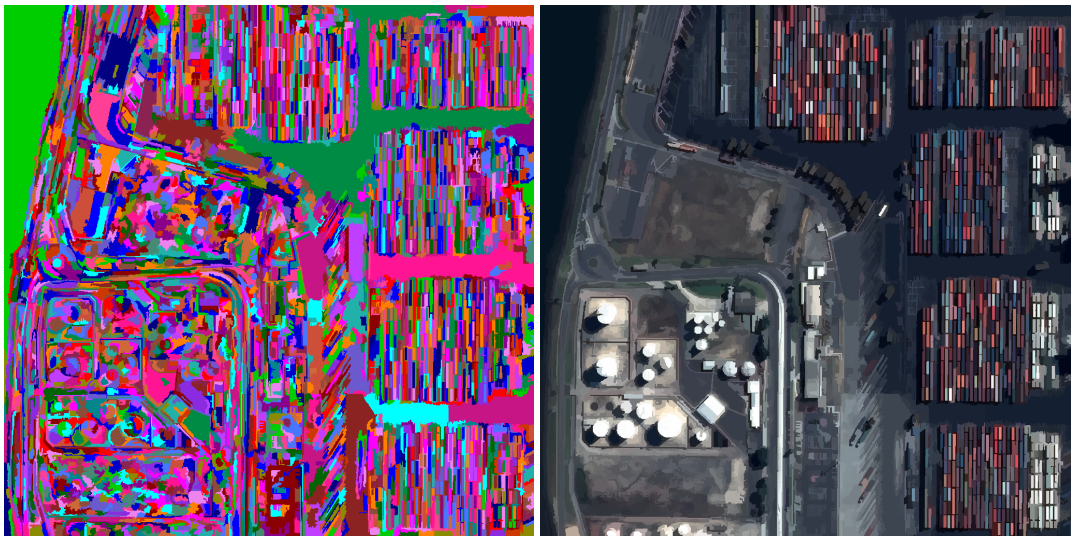
Item 1 Here is the command-line to run the application using the Mean-Shift filter, with default parameters:

```
$ otbcli_Segmentation -in segmentation_small_xt_phr.tif
  -filter meanshift -mode raster
  -mode.raster.out meanshift.tif uint32
```

A better result is obtained by using a spectral radius of 30:

```
$ otbcli_Segmentation -in segmentation_small_xt_phr.tif
  -filter meanshift -filter.meanshift.ranger 30 -mode raster
  -mode.raster.out meanshift.tif uint32
```

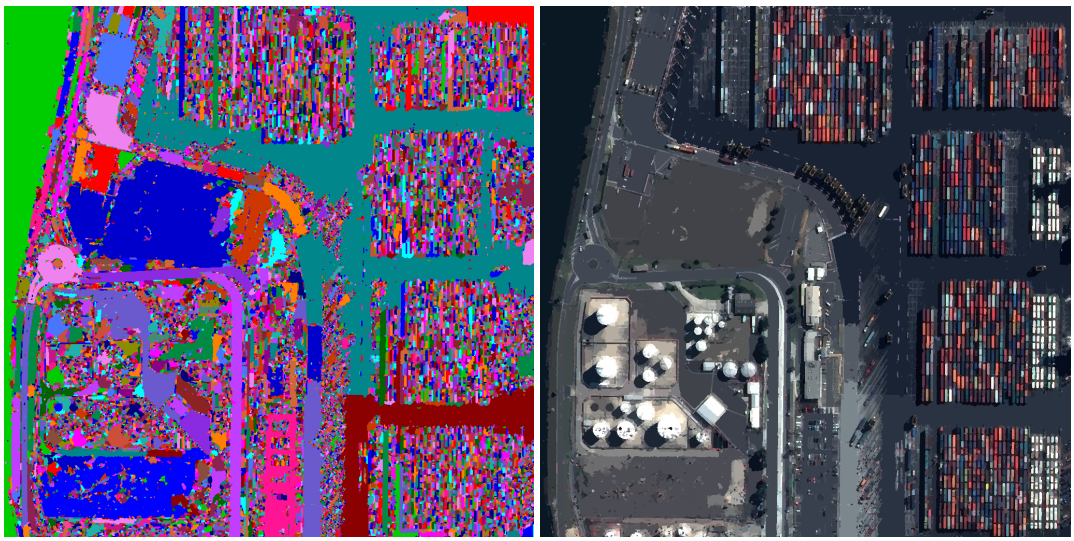
Here are the results of this command:



Here is the command-line to run the application using the Watershed filter, with default parameters:

```
$ otbcli_Segmentation -in segmentation_small_xt_phr.tif
  -filter watershed -mode raster
  -mode.raster.out watershed.tif uint32
```

Here are the results of this command:



3.2.4 Going big: the vector mode

Item 1 The following command will run the application on the larger image:

```
$ otbcli_Segmentation -in segmentation_large_xt_phr.tif
  -filter meanshift -filter.meanshift.ranger 30 -mode raster
  -mode.raster.out meanshift.tif uint32
```

Since the input image is quite large (8192 by 8192 pixels), it is likely that, depending on the available memory on the computer:

- The application fails with a memory allocation error,
- The application does not fails but starts to eat all the available memory.

Item 2 The following command will run the application in *vector* mode, without the *stitch* option:

```
$ otbcli_Segmentation -in segmentation_large_xt_phr.tif
  -filter meanshift -filter.meanshift.ranger 30 -mode vector
  -mode.vector.out meanshift.sqlite -mode.vector.stitch 0
```

In vector mode, the memory consumption is stable because the segmentation on a per tile basis.

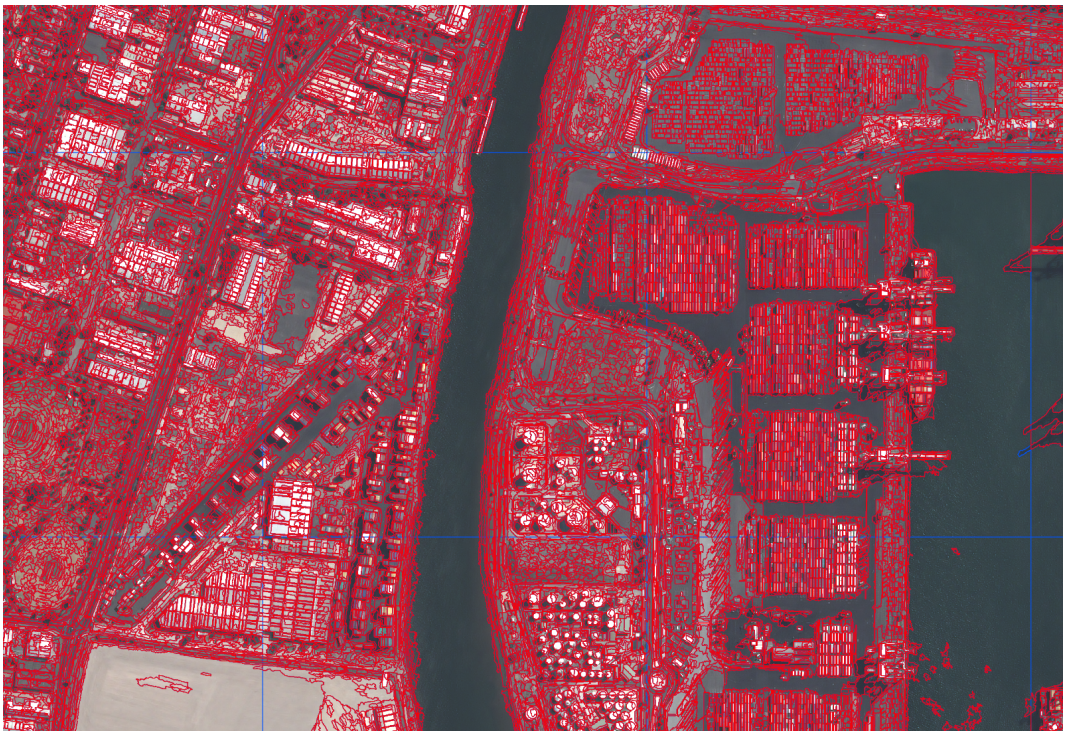
Item 3 In **QGis** we can see the effect of this tile-based segmentation : tiles border are visible in the segmentation result. On can also see that the segmentation produces a large number of polygons.

Item 4 The following command will run the application in *vector* mode, with the *stitch* option enabled:

```
$ otbcli_Segmentation -in segmentation_large_xt_phr.tif
  -filter meanshift -filter.meanshift.ranger 30 -mode vector
  -mode.vector.out meanshift.sqlite -mode.vector.stitch 1
```

Looking at the results in **QGis** one can see that most of the tiling effects have been removed by the stitching option (there might be some left). The results are therefore closer (but not identical) to what we would expect without the tiling strategy.

Here is how the results look like in **QGis**. In blue, one can see the results without stitching, and in red, the results with stitching.



3.2.5 Homework

Item 1

- The *tilesize* option allows to tune the size of the tile used during piecewise segmentation
- The *simplify* option allows to simplify the output polygons up to a given tolerance (always expressed in pixels). The resulting file will be smaller.
- The *minsize* option allows to discard segments whose size is smaller than a user-defined threshold (in pixels).

Item 2 To avoid segmenting vegetation, one can build a vegetation mask with the **BandMath** application by thresholding the NDVI of the image. This mask can then be used in the segmentation application using the *mode.vector.inmask* option. Note that this mode is only available in *vector* mode.

Item 3 Objects with high reflectance values are often more difficult to segment. Because of specular reflections, their inner variance is usually higher than other objects. Therefore, segmentation methods relying on comparison of neighboring pixels with respect to a given threshold will fail (this is the case for all three methods we used during the exercise).

An idea to overcome this issue is to segment together all neighboring pixels with very high reflectance. This can be done with the connected components method, as shown earlier in the solution.

3.3 Learning and classification from pixels

3.3.1 Produce and analyze learning samples

The label corresponding to the **water** class in the shapefile is **2**

3.3.2 Estimate image statistics

Here is the command line to produce the image statistics:

```
$ otbcli_ComputeImagesStatistics
  -il melbourne_ms_toa_ortho_extract_small.tif
  -out melbourne_ms_toa_ortho_extract_small_stats.xml
```

The value of the mean of the red band is **111.625**. Reflectance allows to compare radiometric values between images of different sensors.

3.3.3 Estimate classification model using the Support Vector Machine algorithm

Here is the command line to produce the SVM model:

```
$ otbcli_TrainSVMImagesClassifier
  -io.il melbourne_ms_toa_ortho_extract_small.tif
  -io.imstat melbourne_ms_toa_ortho_extract_small_stats.xml
  -io.vd training.shp
  -io.out melbourne_ms_toa_ortho_extract_small_model.svm
```

Linear kernel is used by default in the application. The measured accuracy is 0.988523. The value is high due to the low number of training samples and their lack of variability.

3.3.4 Apply classification model

Here is the command line to produce the SVM model:

```
$ otbcli_ImageSVMClassifier
  -in melbourne_ms_toa_ortho_extract_small.tif
  -svm melbourne_ms_toa_ortho_extract_small_model.svm
  -imstat melbourne_ms_toa_ortho_extract_small_stats.xml
  -out melbourne_extract_small_classification_5classes.tif
  uint8
```

Pixels of the output image will contain the class label decided by the SVM classifier.

3.3.5 Produce printable classification map

The look-up table (LUT) used to produce the classification map in my case is:

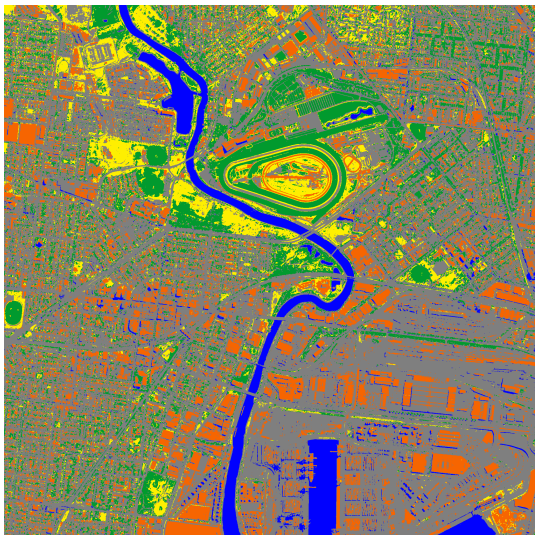
```
# Lines beginning with a # are ignored
#Vegetation (green)
1 1 154 46
#Water (blue)
2 1 1 254
#Soil (yellow)
3 254 239 1
#Roads (grey)
```

```
4 127 127 127
#Buildings (orange)
5 246 101 1
```

Here is the command line to produce the classification map:

```
$ ColorMapping
  -in melbourne_extract_small_classification_5classes.tif
  -out melbourne_extract_small_classification_color.png uint8
  -method custom -method.custom.lut ColorTable.txt
```

And the result of the command:



3.4 Learning and classification from objects

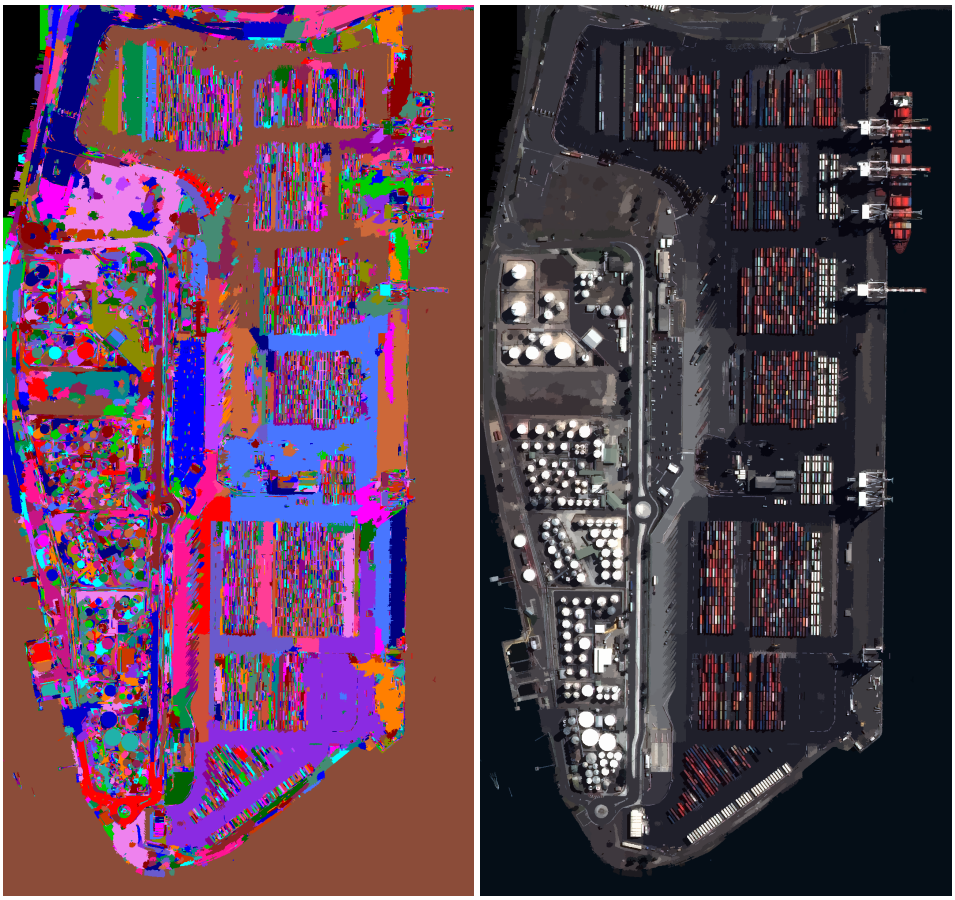
3.4.1 The preliminary segmentation

Item 1 Here are the command-line to generate the color-mapped images:

```
$ otbcli_ColorMapping
  -in phr_pxs_melbourne_xt_small_segmentation.tif
  -out obc_segmentation_optimal.png uint8
  -method optimal

$ otbcli_ColorMapping
  -in phr_pxs_melbourne_xt_small_segmentation.tif
  -out obc_segmentation_image.png uint8
  -method image
  -method.image.in  phr_pxs_melbourne_xt_small.tif
```

Here is what the color-mapped images look like:



Item 2 From the segmentation results, we can infer that an object-based classification method might perform well on:

- Most of circular containers,
- Most of rectangular containers,
- Simple classes like water or roads.

But it will most likely fail on:

- Some circular or rectangular containers that are fragmented by segmentation,
- Complex objects like the boat or the cranes
- Small objects like cars and trucks.

3.4.2 Object Labeling module - basics

Item 3 The *Objects* tab allows to create classes and to add training segments to these classes. The *Features* tab allows to select the object-based features to be used for classification. Last, the *Learning* tab allows to tune classification parameters and to perform the classification.

Item 5 The segment (from the image segmentation) under the mouse pointer gets selected on first right-click action.

Item 6 The selected segment is added to the current class on second right-click action.

Item 10 A SVM classifier is trained according to created classes and corresponding training samples, and the remaining of the image segments are classified using the trained classifier.

Item 11 When the *Save/Quit* button is pressed, the module closes and produces three different outputs:

- An image of labels corresponding to the classes,
- A color-mapped image according to classes colors,
- A vector outputs containing polygons labeled with their predicted classes.

Object Labeling module - advanced In this part of the exercise, we will use these additional files: `samples.xml` and `parameters.xml`

1. Load again the image and the segmentation inside the module.
2. Load the samples file using *File/Load Samples*. What are the different object classes loaded ? How many samples per classes are used ?
3. Load the classification parameters file. What are the features used ?
4. Perform the classification. What are the objects in the image that are badly classified because of missing classes ?
5. What are the objects in the image that are poorly classified because they are badly segmented or too complex ?
6. Try to enhance the classification by adding missing classes.
7. Try to enhance the classification by adding new features.

Tips and Recommendations:

- The **Object Labeling** module is quite memory consuming. Depending on the available memory on your system, you might want to restart **Monteverdi**.

Object Labeling module - active learning

1. In the *Objects* tab, click on the *Sample* button in the lower-left area. This will show you difficult samples by using the margin sampling technique.
2. What kind of segments are considered by the algorithm as hard to classify ?
3. Try to create a *Trash* class to handle noise segments.
4. Perform a few more iteration of active learning. What do you observe ?

3.4.3 Object Labeling module - advanced

Item 2 The classes selected in the samples file are:

- Circular containers
- Rectangular colored containers
- Rectangular white containers
- Water
- Asphalt

Item 4 Using the provided samples and parameters, we get the following result. We can see that some basic classes are detected at the expense of more misclassification on difficult objects, as shown in left part of the figure at the end of this section.

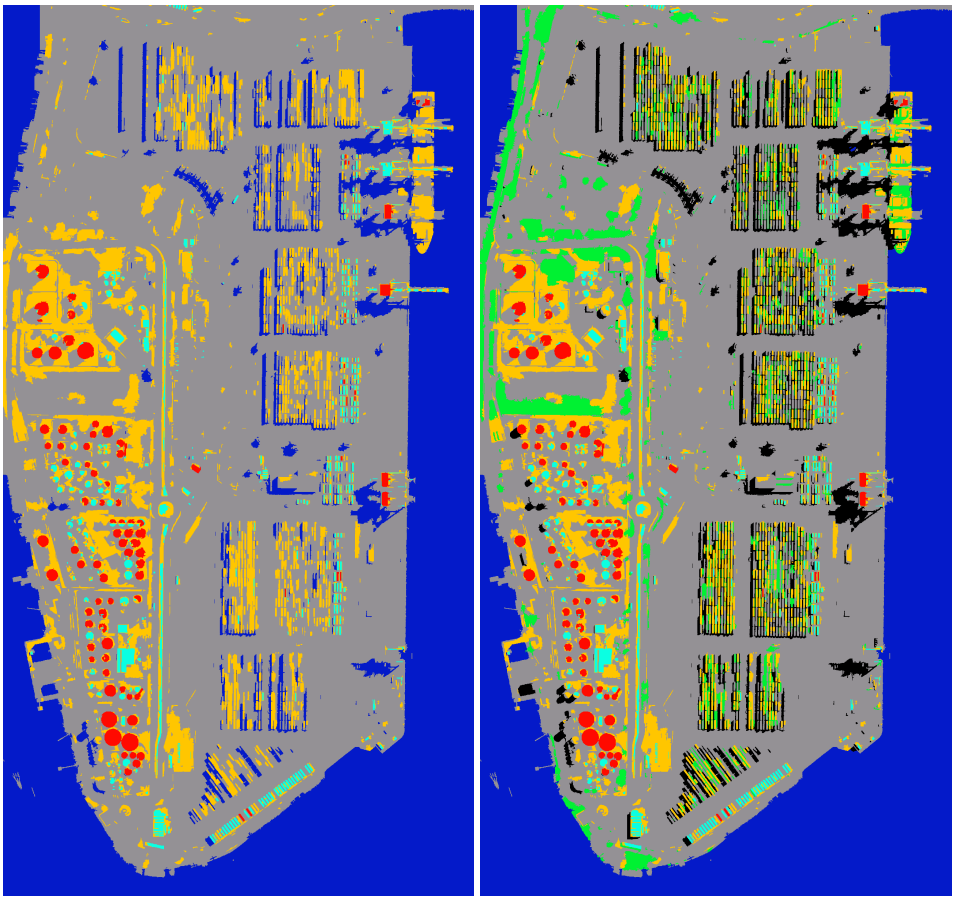
We can see some obvious missing classes in the training set leading to classification errors:

- Shadows area get classified as Water. Even if Shadow is not a strictly-speaking a class of interest, the overall classification quality would benefit from a Shadow class.
- Vegetation areas, even if there are only few of them in the images, also get miss-classified because there is no vegetation class in the training set.

Item 5 As foreseen in section The preliminary segmentation, some objects of interest are poorly segmented or too complex for good classification results:

- The boat and the cranes are too complex,
- Some of the containers (spherical or rectangular) are poorly segmented, which leads to miss-classification.

Item 6 - 7 By adding a few more classes and samples, we get the result presented on the right of following figure.



3.4.4 Object Labeling module - active learning

Item 2 The implemented active learning strategy often shows objects that are difficult to label manually, because they correspond to parts of fragmented objects or to segmentation noise.

Item 4 We can observe that occasionally, the active learning strategy will discover a new kind of object, for which no class has been created yet. It may also run several times into the same objects that are still difficult to classify after some iterations.

3.5 Elevation map from stereo pair

3.5.1 From images to epipolar geometry

Item 1 To get the command-line help, run

```
$ otbcli_StereoRectificationGridGenerator
```

To get the graphical version of the **StereoRectificationGridGenerator** application, run

```
$ otbgui_StereoRectificationGridGenerator
```

Item 2 The application estimates the displacements to apply to each pixel in both input images to obtain epipolar geometry.

Item 3 The **GridBasedImageResampling** application allows to resample the two input images in the epipolar geometry using these grids. These grids are intermediary results, not really useful on their own in most cases.

```
$ otbcli_StereoRectificationGridGenerator
  -io.inleft tristereo_melbourne_3_small_ref.tif
  -io.inright tristereo_melbourne_1_small_ref.tif
  -io.outleft 31_grid_tristereo_melbourne_3_small_ref.tif
  -io.outright 31_grid_tristereo_melbourne_1_small_ref.tif
  -epi.elevation avg -epi.elevation.avg.value 20.45
```

Item 4 For the left image :

```
$ otbcli_GridBasedImageResampling
  -io.in tristereo_melbourne_3_small_ref.tif
  -io.out 31_epi_tristereo_melbourne_3_small_ref.tif
  -grid.in 31_grid_tristereo_melbourne_3_small_ref.tif
  -out.sizex 1237 -out.sizey 1237
```

For the right image:

```
$ otbcli_GridBasedImageResampling
  -io.in tristereo_melbourne_1_small_ref.tif
  -io.out 31_epi_tristereo_melbourne_1_small_ref.tif
  -grid.in 31_grid_tristereo_melbourne_1_small_ref.tif
  -out.sizex 1237 -out.sizey 1237
```

3.5.2 Refinement of epipolar geometry

Item 1 The epipolar couple generated with the images with refined geometry does not present disparities in the vertical direction.

Item 2 Here is the command-line to run the **ConcatenateImages** application to generate the anaglyph image:

```
$ otbcli_ConcatenateImages -il
  31_epi_tristereo_melbourne_3_small_ref.tif
  31_epi_tristereo_melbourne_1_small_ref.tif
  31_epi_tristereo_melbourne_1_small_ref.tif
  -out 31_anaglyph_3_1.tif
```

Here is the result of this command:

3.5.3 Block matching

Item 1 The mandatory parameters are the intervals of disparity in the horizontal and vertical direction. In our case the interval in vertical direction should be void.

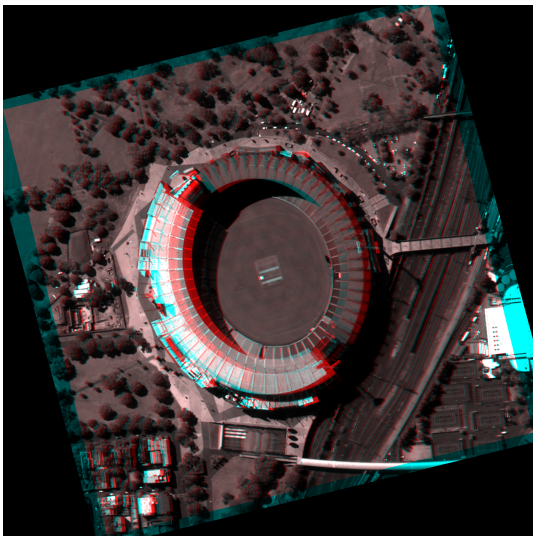


Figure 2: Epipolar geometry

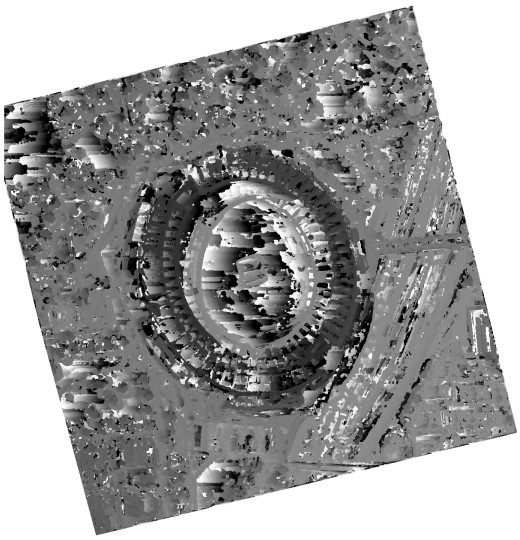
Item 2 In theory, the block matching can perform a blind exploration and search for a infinite range of disparities between the images of the stereo pair. We need now to evaluate a range of disparities where the block matching will be performed.

In our case, we take one point on a *ground* area. The image coordinate in the first image is is $[275,343]$ and in the second image is $[277,343]$. We then select a second point on a higher region (in our case a point near the top of the Melbourne Cricket Ground) The image coordinate of this pixel in the first image is $[712,354]$ and in the second image is $[671,354]$. We can see that for the horizontal exploration, we must set the minimum value lower than -41 and the maximum value higher than 2 (pay attention to the convention for the sign of the disparity, the range is defined from the left to the right image).

Item 3 Here is the command-line to run the application with default parameters:

```
$ otbcli_BlockMatching
  -io.inleft 31_epi_tristereo_melbourne_3_small_ref.tif
  -io.inright 31_epi_tristereo_melbourne_1_small_ref.tif
  -io.out 31_disparity_map_3_1.tif
  -bm.minhd -40 -bm.maxhd 40 -bm.minvd 0 -bm.maxvd 0
```

and here the result of this command:



It shows that we need to discard pixels where block matching does not work and also filter low correlation values.

3.5.4 Advanced Block matching: refinement of the disparity map

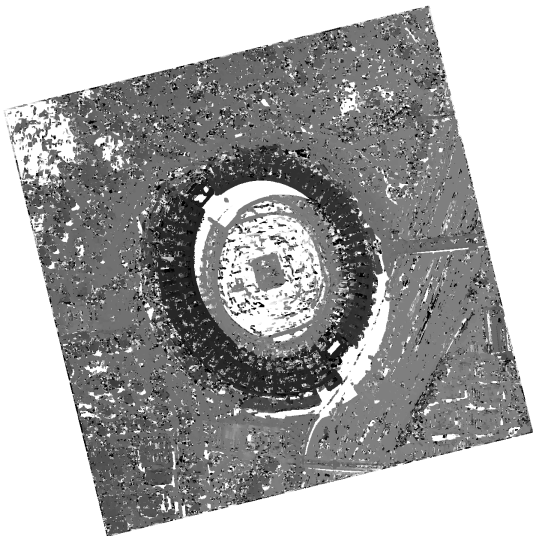
Item1 Use the following parameters: **-io.outmetric 1 -bm.metric ncc**

Item2 Use the **mask.variancet** parameter.

Here is the command-line to run the application witch combine all these parameters:

```
$ otbcli_BlockMatching
  -io.inleft 31_epi_tristereo_melbourne_3_small_ref.tif
  -io.inright 31_epi_tristereo_melbourne_1_small_ref.tif
  -io.out 31_disparity_map_3_1.tif
  -bm.minhd -40 -bm.maxhd 40 -bm.minvd 0 -bm.maxvd 0
  -mask.nodata 0 -mask.variancet 100 -io.outmetric 1
  -bm.metric ncc
```

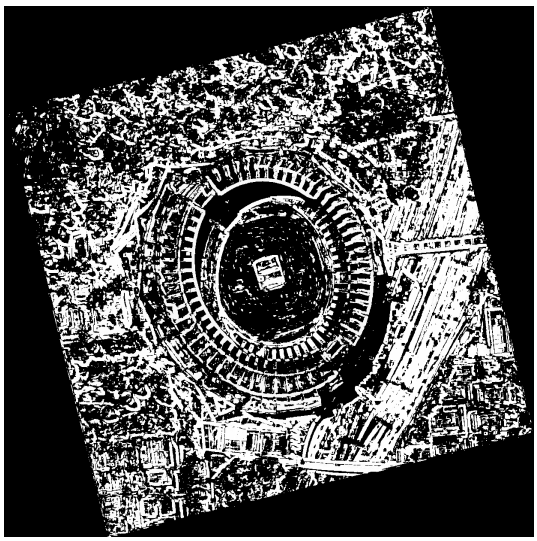
Here is the result of this command:



Item3

```
$ otbcli_BandMath
  -il 31_disparity_map_3_1.tif
  -out 31_filtered_disparity_map_3_1.tif
  -exp "if(im1b3>0.9,im1b1,-1000)"
```

Here is the result of this command:



3.5.5 From disparity map to ground elevation

Item1 Here is the command-line to run the application:

```
$ otbcli_DisparityMapToElevationMap
  -io.in 31_disparity_map_3_1.tif
  -io.left tristereo_melbourne_3_small_ref.tif
  -io.right tristereo_melbourne_1_small_ref.tif
  -io.lgrid 31_grid_tristereo_melbourne_3_small_ref.tif
  -io.rgrid 31_grid_tristereo_melbourne_1_small_ref.tif
  -hmin 0 -hmax 80 -elev average -step 1
  -elev.average.value 20.45
  -io.out 31_disparity_map_to_elevation_3_1.tif
```

Here is the result of the command:

Item2 I found 20 meters for the ground and 58m for the roof. See this Wikipedia article for ground truth.