TeraTools: enabling Ultra-Terabyte Image Processing

Alessandro Bria, Department of Electrical and Information Engineering, University of Cassino and Southern Lazio, a.bria@unicas.it
Giulio Iannello, Department of Engineering, University Campus Bio-Medico di Roma, g.iannello@unicampus.it

TeraTools is a suite of software tools designed to process ultra-terabyte sized image datasets. These datasets may contain both very large 4D images (three spatial dimensions plus multiple channels) and time series of relatively small 4D images. TeraTools are cross-platform and are currently available on MacOS, Linux and Windows.

TeraTools are open-source and available at http://abria.github.io/TeraStitcher/.

Some tools of the suite are relatively stable and are available within the installer. Other tools are still under testing and are included in the distribution as source code or are not included yet. All features described in the following are included in the installer, unless differently stated with the words: "*source code only*" or *"not included"*.

Currently the suite covers the following functionalities:
1. *Stitching*: the computation of the correct alignments of the tiles in which a large volume has been partitioned during acquisition and the generation of a stitched image. Available both through a command line interface and a GUI.
2. *Format conversion*: the conversion of an already stitched image to a different format more suited for processing or visualization. Available through a command line interface only.
3. *Parallel execution of stitching and conversion*: parallelism available at hardware level can be exploited to speedup both stitching and conversion of very large images. It requires python, an MPI installation, and the mpi4py package. **Not included**.
4. *Transparent access*: the efficient extraction of any sub-region of a larger image for further processing (image enhancement, object detection and/or segmentation, etc.). Available through a python interface. **Not Included**.
5. *Visualization and annotation*: the ability to explore and annotate the image in 3D. **Not included**, but it is available within the Vaa3D software (www.vaa3d.org) under the menu "Advanced > Big Image Data > TeraFly"..

In the following the functionalities of the TeraTools are presented making reference to their command line interface, since this interface gives full access to the options available.

To Download/use/run/edit/change any portion of the code of the suite users must agree to the following license.

****
1. This material is free for non-profit research, but needs a special license for any commercial purpose. Please contact Alessandro Bria at a.bria@unicas.it or Giulio Iannello at g.iannello@unicampus.it for further details.
2. You agree to appropriately cite this work in your related studies and publications (see below for references).
3. This material is provided by the copyright holders (Alessandro Bria and Giulio Iannello), University Campus Bio-Medico and contributors "as is" and any express or implied warranties, including, but not limited to, any implied warranties of merchantability, non-infringement, or fitness for a particular purpose are disclaimed. In no event shall the

copyright owners, University Campus Bio-Medico, or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; reasonable royalties; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

4. Neither the name of University Campus Bio-Medico of Rome, nor Alessandro Bria and Giulio Iannello, may be used to endorse or promote products derived from this software without specific prior written permission.

****

If use our tools in your work, please cite one or more of the following publications where you can find additional information with respect to this guide:

[1] Bria A, Iannello G , "TeraStitcher - A tool for fast automatic 3D-stitching of teravoxel-sized microscopy images", BMC Bioinformatics, 2012, 13:316

[2] Bria A, Iannello G, Peng H, "An open-source Vaa3D plugin for real-time 3D visualization of Terabyte-sized volumetric images", ISBI 2015, April 16, 2015

[3] Bria A, Iannello G, Onofri L, Peng H, "TeraFly: real-time three-dimensional visualization and annotation of terabytes of multidimensional volumetric images", Nature Methods, February 25, 2016

[4] Bria A, Bernaschi M, Guarrasi M, Iannello G, "Exploiting multi-level parallelism for stitching very large microscopy images", Frontiers in Neuroinformatics, 13, art. no. 41, 2019.

1 Stitching

The first tool of the suite is TeraStitcher, designed for computing the correct alignments of the tiles in which a large volume has been partitioned during acquisition.
TeraStitcher assumes that the *Raw Unstitched Image*, i.e. what is generated by the acquisition system (RUI in the following), consists of overlapping tiles arranged according to a 2D regular matrix along X-Y dimensions. Each tile consists of 2D slices along the Z dimension. All slices must have the same size along X-Y dimensions.
Tile slices can be stored as either series of 2D slices, or as one or more 3D sub-stacks. The first alternative is referred to as "TiledXY|2Dseries", whereas the second one is referred to as "TiledXY|3Dseries".
RUIs can be sparse, which means that tiles can be incomplete, i.e. not all tiles must have the same number of slices and they may even be empty.
The RUI must be imported, which means that an xml file (the *import xml file* in the following) must be produced that is compliant with what TeraStitcher expects (see below for details).
After the xml import file has been generated, all other steps of the stitching process can be executed on any dataset, providing that the dataset meets the above assumptions and that image data are stored in a supported input file format.

TeraStitcher performs stitching in six steps:
- Step 1 (import): an xml import file containing the relevant information about the RUI is generated.
- Step 2 (displcompute): multiple relative displacements (i.e. alignments) between each pair of tiles that are adjacent in the tile matrix are computed with a reliability score.
- Step 3 (displproj): for each pair of adjacent tiles the displacement with the best reliability score is chosen.
- Step 4 (displthresh): displacements with too low reliability (less than a threshold) are discarded
- Step 5 (placetiles): a global optimization algorithm is applied to determine a coherent set of displacements between adjacent tiles.
- Step 6 (merge): a multi-resolution stitched (i.e. without overlapping regions) image is generated.

Step 6 can be performed using TeraConverter, a different tool that can convert images in a number of supported format and that can accept as an input format also the output of any step of TeraStitcher, i.e. a RUI and its alignment information.

Early versions of TeraStitcher had their own merge step that was less general than the function provided by TeraConverter. For this reason, the merge step of TeraStitcher is not supported any longer and it will not be further documented here.

Besides the command line interface of both TeraStitcher and TeraConverter, a GUI interface that integrates both TeraStitcher for steps 1-5 and TeraConverter for step 6 is included in the installer. A user guide for this GUI is in preparation.

1.1 TeraStitcher input formats

TeraStitcher can accept and generate images in several formats.

In most cases images are read using special modules referred to in the following as *input plugins* and characterized by a well defined interface. Additional input plugins supporting new formats can be added providing they implement this interface.
Input plugins can be either 2D or 3D plugins according to they are designed to support 2D of 3D file formats. Moreover they are characterized as either *non-interleaved* or *interleaved* according to the supported format represents channels in spatially separate planes or not. In the table below are reported all the plugins currently available with their characteristics and limitations.

Name	2D/3D	Format description	Interleaved	Notes
opencv2D	TiledXY|2Dseries	Any 2D format supported by the OpenCV library (tested with rel. 2.4.8)	yes	Tested with Rel. 2.4.8 http://opencv.org/opencv-2-4-8.html
tiff2D	TiledXY|2Dseries	Standard TIFF	yes	Files larger that 4 GB allowed (BigTiff format) http://www.awaresystems.be/imaging/tiff.html
tiff3D	TiledXY|3Dseries	Standard multi-page TIFF	yes	Files larger that 4 GB allowed (BigTiff format) http://www.awaresystems.be/imaging/tiff.html
IMS_HDF5	TiledXY|3Dseries	Imaris File Format (Imaris 5.5) See section *Processing Imaris datasets* for details	no	HDF5-based format http://open.bitplane.com/Default.aspx?tabid=268 Compatible with *Fusion* http://www.andor.com/microscopy-systems/dragonfly
Bioformats2D	TiledXY|2Dseries	Any 2D image accepted by the Bio-Formats library	no	Partially tested with bioformats_package.jar, Rel. 5.0.3 http://downloads.openmicroscopy.org/bio-formats/5.0.3/ See **Appendix E** for how to use this plugin
Bioformats3D	TiledXY|3Dseries	Any image accepted by the Bio-Formats library	no	Partially tested with bioformats_package.jar, Rel. 5.0.3 http://downloads.openmicroscopy.org/bio-formats/5.0.3/ See **Appendix E** for how to use this plugin
magellan	TiledXY|3Dseries	Micro-Magellan file format (Image File Stack)	no	Partially tested https://micro-manager.org/wiki/Micro-Manager_File_Formats

WARNING: bioformats and magellan plugins are currently NOT included in the distribution.

1.2 Import step

1.2.1 The standard import step

The *standard* import step of TeraStitcher assumes that the RUI is stored according to the organization described at:

https://github.com/abria/TeraStitcher/wiki/Supported-volume-formats#two-level-hierarchy-of-folders

To execute the standard import step TeraStitcher must be given:
- The path where the two-level hierarchy of folders is stored (command option –volin)
- The method used to store tile slices (command option –volin_plugin)
- The voxel size in each dimension in microns (command options –vxl1 –vxl2 –vxl3)
- The physical coordinates system used by the acquisition system to name the folders in the two-level hierarchy of folders (command options –ref1 –ref2 –ref3, see section *The reference system* for deatils)

Other relevant command options for the import step (optional) are:
- --imin_channel: it is one of the letters 'R', 'G' or 'B' or a digit: '0', '1', … , '9'[1]; it must be specified if the image is multi-channel (any channel is equivalent in the import step, but the most informative should be chosen to generate a useful middle test image)
- --imout_depth: it is the color depth in bits of the output image; it must be specified if the middle slice test image generated during the import phase must be saved with a color depth of 16 bits
- --imin_plugin: file format to be used to read the individual files in the RUI; it must be specified if the format file is not TIFF
- --sparse_data: it is a flag; it must by specified if the RUI is sparse

The xml file generated by the import step of TeraStitcher is the only information needed to perform all other steps of TeraStitcher as well as to generate the stitched image by means of TeraConverter.

If the RUI can be saved during acquisition (or easily re-organized after acquisition) according to the two-level hierarchy of folders mentioned above, the standard import step of TeraStitcher can be used to generate the import xml file.

More details about the xml import file can be found at:

https://github.com/abria/TeraStitcher/wiki/Supported-volume-formats#xml-descriptors

1.2.2 The import step using an externally generated xml import file

Alternatively, an independently developed tool can be used to generate the xml import file. Thanks to the ability of TeraStitcher to use regular expressions to discriminate the names of files containing slices belonging to a given tile, any folder organization can be used to save the RUI. Nevertheless, to generate the xml import file (see link above), at least the following information should be made available in some way:
- the kind of tile matrix (2D or 3D[2])
- the folder where the RUI is stored
- the orientation of the reference system in which tile positions are given (see more details below).

[1] See section for details and in particular when use letters and when use digits.
[2] For 3D tiled volumes see the section *Importing 3D tiled RUIs* below.

- the voxel size
- the number of channels
- the number of bytes per channel
- the dimensions of the tile matrix
- the paths of files storing the slices of each tile (different tiles may have the same path)
- how file names storing the slices of each tile can be selected using regular expressions (this information is needed only if files storing the slices of different tiles have the same path)
- either the nominal positions of the tiles in the physical space of the acquisition instrument or, alternatively, the mechanical displacements between tiles in X-Y dimensions

If the RUI is sparse the generation of the xml import file is more complex and more information is needed, depending on the way tile slices are stored (if as 2D images or as 3D substacks). See Z_RANGES attribute in the xml import file documented at the link above for more information.

A simple way to generate an xml import file starting from the above listed information is to generate a text file containing all of them. To simplify this task for those that cannot rearrange their RUIs according to the folder structure expected by TeraStitcher, a few python scripts are available. These scripts are:

- **`import_objects.py`** containing the classes `ImportData` and `StackDescriptor` used to represent in a structured way the above information about the RUI
- **`gen_xmlimport.py`** containing the code that generates a provisional xml import file
- **`parse_ini_file.py`** containing the code that creates an ImportData object parsing and .ini file containing the information about the RUI in a format inspired to the windows .ini files[3]; if the information about RUI organization is available in other forms, this python script can be substituted by any other parsing algorithm providing that it contains a function returning an object of class `ImportData` initialized with the right import information (see comments in classes `ImportData` e `StackDescriptor` for more details).

The first two scripts are independent of the format used to generate the text file. The third script can be used either to read directly an .ini file if the format described in the section *Writing a* .ini *file describing the RUI configuration* can be easily generated by the software controlling the acquisition system, or as an example of how the data structure required by the first script can be generated.

If these scripts are used, a provisional import xml file is created that can be accepted by TeraStitcher and TeraConverter.

A final note is that, as it has been pointed out, this solution currently cannot deal with sparse datasets, i.e. datasets in which some tiles are missing or incomplete. For these datasets the folder hierarchy and the file naming conventions used for the standard import step are still the only viable option.

[3] The format of this .ini file is described in detail in the section *Writing a* .ini *file describing the RUI configuration*.

1.3 The reference system

In the import step, when the two-level hierarchy of folders is used to specify the dataset structure, the reference system used at acquisition time must be specified using the command options --ref1, --ref2 and --ref3, according to the following conventions. These conventions must be followed also if an externally generated xml import file is used, if the association of the image to a physical acquisition space must be kept coherent.

The physical acquisition space is assumed to have a Cartesian reference system. The two dimensions of the X-Y plane are conventionally named as 'V' the vertical one, also commonly referred to as 'Y', and as 'H' the horizontal one, also commonly referred to as 'X'.
The third dimension, commonly referred to as 'Z', is conventionally named as 'D', for depth. These three dimensions are assumed ordered, being V the first dimension, H the second, and D the third. Indeed, if we see a 3D image as a series of slices, and each slice as a 2D matrix of voxels, the first index V is the index of rows of slices, the second index H is the index of columns of slices, and the third index D is the slice index.
When tiles are saved during acquisition, they may have a nominal physical position in the reference space, i.e. they are associated to three real numbers that specify the position of some fixed point of the tile, usually the left-most, upper vertex of the first slice. The third number is assumed to be always the coordinate along dimension D, but the first two may refer to either V and H, or to H and V, respectively, depending on the orientation of the pixel matrix with respect to the physical reference system. In the first case, the first coordinate refers to the first dimension and the second coordinate to the second dimension of the voxel matrix, whereas in the second case the opposite happens. Moreover, considering the relations between physical coordinates and indices in the 3D image matrix, for any dimension physical coordinates can increase or decrease with the corresponding index.
To precisely specify the relations between physical acquisition space and indices in the 3D voxel matrix we use the command options --ref1, --ref2 and --ref3, associated respectively to dimensions V, H, and D, to specify which is the correspondent coordinate provided by the acquisition system. A positive number is then used to associate a coordinate with a dimension if the coordinate grows with increasing indices, whereas a negative number is used in the opposite case.

For instance the command options

--ref1=1 --=2 --ref3=3

mean that the first coordinate refers to V, the second to H and the third to D, and that all grow with increasing indices. Conversely the command options:

--ref1=2 --ref2=-1 --ref3=3

mean that the first coordinate refers to H, the second to V and the third to D, and that coordinates along H decrease with increasing indices.

With this convention, assuming that the third coordinate (the one identifying the slice) is always associated to D, there are 16 different possible reference systems, which are listed in Table I. It is worth noting however that TeraStitcher currently does not support reference systems in which coordinates along D decrease with increasing indices.

Ref1	ref2	ref3	Description
1	2	3	First coordinate is associate to V and increases with indices Second coordinate is associated to H and increases with indices Third coordinate is associated to D and increases with indices
-1	2	3	First coordinate is associate to V and decreases with indices Second coordinate is associated to H and increases with indices Third coordinate is associated to D and increases with indices
1	-2	3	First coordinate is associate to V and increases with indices Second coordinate is associated to H and decreases with indices Third coordinate is associated to D and increases with indices
-1	-2	3	First coordinate is associate to V and decreases with indices Second coordinate is associated to H and decreases with indices Third coordinate is associated to D and increases with indices
2	1	3	First coordinate is associate to H and increases with indices Second coordinate is associated to V and increases with indices Third coordinate is associated to D and increases with indices
-2	1	3	First coordinate is associate to H and decreases with indices Second coordinate is associated to V and increases with indices Third coordinate is associated to D and increases with indices
2	-1	3	First coordinate is associate to H and increases with indices Second coordinate is associated to V and decreases with indices Third coordinate is associated to D and increases with indices
-2	-1	3	First coordinate is associate to H and decreases with indices Second coordinate is associated to V and decreases with indices Third coordinate is associated to D and increases with indices
1	2	-3	First coordinate is associate to V and increases with indices Second coordinate is associated to H and increases with indices Third coordinate is associated to D and decreases with indices
-1	2	-3	First coordinate is associate to V and decreases with indices Second coordinate is associated to H and increases with indices Third coordinate is associated to D and decreases with indices
1	-2	-3	First coordinate is associate to V and increases with indices Second coordinate is associated to H and decreases with indices Third coordinate is associated to D and decreases with indices
-1	-2	-3	First coordinate is associate to V and decreases with indices Second coordinate is associated to H and decreases with indices Third coordinate is associated to D and decreases with indices
2	1	-3	First coordinate is associate to H and increases with indices Second coordinate is associated to V and increases with indices Third coordinate is associated to D and decreases with indices
-2	1	-3	First coordinate is associate to H and decreases with indices Second coordinate is associated to V and increases with indices Third coordinate is associated to D and decreases with indices
2	-1	-3	First coordinate is associate to H and increases with indices Second coordinate is associated to V and decreases with indices Third coordinate is associated to D and decreases with indices
-2	-1	-3	First coordinate is associate to H and decreases with indices Second coordinate is associated to V and decreases with indices Third coordinate is associated to D and decreases with indices

Table I: The 16 possible reference systems for the acquisition instrument space (the gray cells correspond to reference system that are not currently supported by TeraStitcher).

As mentioned above, the actual reference system used by the motorized microscope must be specified when the import step is executed relying on the two-level hierarchy of folders, otherwise the dataset structure is erroneously imported, and tiles are stitched in flipped positions. Conversely, when an externally generated xml import file is used, command line

options --ref1, --ref2 and --ref3 are ignored and the reference system specified in the xml file is taken as valid. In the latter case, each dimension of the voxel size must have the same sign of the corresponding axis in the reference system. For instance, if reference system is:

--ref1=2 --ref2=-1 --ref3=3

the voxel size along H must be negative.

1.4 Alignment computation

The xml import file already contains alignment information that can be used to generate a stitched image. These alignments are the nominal ones, i.e. those generated by the motorized acquisition system.
If nominal alignments are not reliable and they can introduce significant artifacts in the stitched image, alignment errors have to be corrected using steps 2-5 of TeraStitcher. Each step generates an xml file containing improved alignment information. However, only the xml generated by step 5 (*global optimization for tile placement*) contains the corrected alignments that can be used to generate a stitched image without artifacts.

The details of the algorithms used to compute the correct alignments between adjacent tiles are described in reference [1]. Here we report in the following table the parameters that influence the behavior of these algorithms.

Parameter	Command line option	Notes
Input channel	--imin_channel=*chanID*	chanID is a either letter 'R', 'G', or 'B', or a digit '0', '1', … , '9'. In step 2 of TeraStitcher is the channel used to compute the alignments. Significant only for multi-channel datasets.
Substack depth	--subvoldim=*depth*	*depth* is a positive integer. In step 2 of TeraStitcher adjacent tiles are partitioned in substacks of up to *depth* slices. A separate alignment with its reliability is computed for each pair of corresponding substacks. The most reliable alignment is retained by step 3 of TeraStitcher.
Search range in V dimension	--sV=*range*	*range* is a non negative integer. In step 2 of TeraStitcher, starting from the nominal alignment a better alignment is searched in V dimension in the range [-*range*, *range*].
Search range in H dimension	--sH=*range*	*range* is a non negative integer. In step 2 of TeraStitcher, starting from the nominal alignment a better alignment is searched in H dimension in the range [-*range*, *range*].

Search range in D dimension	--sD=*range*	*range* is a non negative integer. In step 2 of TeraStitcher, starting from the nominal alignment a better alignment is searched in D dimension in the range [-*range*, *range*].
Threshold to eliminate unreliable alignments	--threshold=*value*	*value* is a real value in [0,1]. In step 4 of TeraStiticher all alignments which reliability is less than value are discarded.

It is worth noting that the value of these parameters must be adapted to the characteristics of the source dataset. More specifically:

- *Substack depth* should be sufficiently large to include with high probability in the substacks information (i.e. marked objects) useful for alignment.
- *Search ranges* should be larger than expected errors introduced by the motorized acquisition system. Consider that they are expressed in voxels while errors are typically expressed in distance units (e.g. um). Hence, these ranges should increase if voxel sizes reduce.
- The *threshold* should be chosen taking into account the sharpness of the objects used for alignment computation. Although the threshold should filter away unreliable alignments that could introduce alignment artifacts, if input images contain relatively smooth objects, lower thresholds should be chosen to avoid discarding too many alignments.

Consider that step 2 of TeraStitcher can be computationally intensive for very large images. Workload slightly increases with *substack depth*, whereas it greatly depend on *search ranges*. Since resolution is typically higher in V-H dimensions, workload is generally dominated by the values of sV and sH options and it roughly grows linearly with their product.

In a version of TeraStitcher that has not been released yet, A CUDA version of the MIP-NCC algorithm has been included. See Appendix A for more details.

After pairwise alignments between adjacent tiles have been computed (step 2 of the pipeline), if the substack depth used is less than the total number of slices of the tiles, multiple alignments per tiles pair are available. Using the reliability associated to each alignment, one alignment per adjacent tile pair is chosen (step 3 or *displacement projection*), and alignments with too low reliability are discarded (step 4 or *thresholding*). This leads to have just one (sufficiently reliable) alignment between each adjacent tile pair. If there is no reliable alignment between an adjacent tile pair, the nominal alignment with reliability 0 is chosen. Note that these steps are carried out separately for each dimension (V, H and D) of the image.

1.5 Global optimization of tile placement

Once a sufficiently reliable displacement or, alternatively, the nominal alignments with reliability 0 has been assigned to each adjacent tile pair, all alignments have to be made compatible before to proceed to the final *merge* step (step 6).
Indeed, considering each group of four adjacent tiles forming a square, the alignments between them are compatible if their sum is 0. More formally, if the tile matrix is modeled as a labeled graph where every edge between adjacent tiles is labeled with the displacement

between them, the set of alignments is compatible if the sum of labels over the edges forming any cycle is 0.

For this reason, the computed alignments (including the nominal ones when the reliability was too low) have to be "adjusted" to meet the above introduced *compatibility* condition.

This can be performed by means of a global optimization that tries to find a set of compatible alignments that minimizes the differences with the computed alignments, taking into account their reliability.

Currently two different global optimization algorithms have been implemented. Both consider the tile matrix as a labeled graph, with the tiles as vertices and the edge between each pair of adjacent tiles labeled with the inverse of the reliability of the computed alignment between them.

The first algorithm, referred to as MST (which stands for *Minimal Spanning Tree*), is based on finding the minimal path between each vertex of the graph (i.e. each tile) and another tile chosen as a reference. This algorithm tries to preserve the alignments with the highest reliability and to introduce changes, that can be also relevant in less reliable, or even no reliable at all, alignments. This algorithm is suited to be used with relatively small tile matrices and/or when alignments are either highly reliable or not reliable at all. MST is the default algorithm and it can be always used.

The second algorithm, referred to as LQP (which stands for Linear Quadratic Programming), is based on minimizing the differences between the computed alignments and the compatible ones, weighted with the inverse of displacement reliabilities. This algorithm may change in principle all alignments, but taking into account their reliability, and keeping all changes as small as possible. It is suited when the tile matrix is large and path towards the reference tile may be long and/or when most alignments are more or less reliable. LQP can be used specifying the command line option `--algorithm=LQP`, but it requires that a python interpreter and packages *numpy* and *scipy* are available, as well as the environment variable `__LQP_PATH__` be properly set (see Appendix B for more details).

1.6 Stitched image generation

The stitched image can be generated using TeraConverter giving in input the RUI and its alignment information. Currently TeraConverter is integrated in the TeraStitcher GUI, but its command line interface has to be used to have full access to its options. See section *TeraConverter* for more details.

When TeraConverter is used to generated a stitched image from a RUI and the alignment information generated by TeraStitcher, the following command line options have to specified:

- --sfmt="TIFF (unstitched, 3D)";
- --imin_plugin=*the plugin name corresponding to the format in which the tile of the RUI are stored* (see section *TeraStitcher input formats*);
- -s=*the complete path of the xml file generated by any step of TeraStitcher*[4];

[4] At any step TeraStitcher generates an xml file containing valid alignment information. After steps 1-4 the alignments are the nominal ones (i.e. those provided at the import step),

- --dfmt=*one of the available output format name* (see table below);
- -d= *the complete path of the destination folder/file* (see table below).

The TeraConverter can then generate the stitched images in several output formats reported in the table below. Note that no output plugins are used by TeraConverter for image generation.

1.6.1 Generation of TIFF images

If the chosen output format saves the image data as TIFF files, a few options are available to control the characteristics of the generated image. First images can be generated in a lossless compressed format or, alternatively, in an uncompressed format. In the latter case, one of more lines of each slice can be packed in one *strip*. Packing more lines in the one strip increases the compression factor, makes faster the access to the whole slice, but may greatly increase to sub-regions of slices. Finally, images may be generated using the BigTiff format that supports single TIFF files larger than 4 Gbytes. The characteristics of the generated TIFF files are controlled by the command line options of TeraConverter.

Format name	**Description**	**Destination (folder/file)**	**Max #channels**	**Notes**
TIFF (series, 2D)[1]	A folder containing a series (1+) of 2D TIFF files (grayscale or RGB)	folder	3 (RGB)	Compressed (default) / uncompressed. Each image slice is a unique file. Access to subregions in x-y inefficient for very large slices.
TIFF (tiled, 2D) [1]	y-x-z hierarchy of separate tiles, each tile is a series of 2D TIFF files (grayscale or RGB)	folder	3 (RGB)	Compressed (default) / uncompressed. Efficient access to subregions. Potentially millions of files for very large images.
TIFF (tiled, 3D) [1]	y-x-z hierarchy of tiles, each tile is a series of multipage 3D TIFF files (grayscale or RGB)	folder	3 (RGB)	Compressed (default) / uncompressed. Efficient access to subregions.
TIFF (tiled, 4D) [1]	c-y-x-z hierarchy of tiles, each tile is a series of single-channel multipage 3D TIFF files (grayscale only)	folder	unlimited	Compressed (default) / uncompressed. Efficient access to subregions. Efficient access to subsets of channels.
HDF5(BigDataViewer)	A HDF5 file containing a BigDataViewer 3D image (grayscale or multi-channel)	file	unlimited	Compressed. Efficient access to subregions. Efficient access to subsets of channels.

whereas after step 5 they are the result of the correct alignment computation (step 2) and the global optimization algorithm (steps 3-5).

HDF5(Imaris IMS)	A HDF5 file containing a Imaris 3D image (grayscale or multi-channel)	file	unlimited	Compressed. Efficient access to subregions. Efficient access to subsets of channels. See section *Processing Imaris datasets* for details
Vaa3D raw (series, 2D)	A folder containing a series (1+) of 2D Vaa3D raw files (grayscale or multi-channel)	folder	unlimited	Uncompressed. Each image slice is a unique file. Access to subregions in x-y inefficient for very large slices.
Vaa3D raw (tiled, 3D)	y-x-z hierarchy of tiles, each tile is a series of 2D Vaa3D raw files (grayscale or multi-channel)	folder	unlimited	Uncompressed. Efficient access to subregions.
Vaa3D raw (tiled, 4D)	c-y-x-z hierarchy of tiles, each tile is a series of 3D single-channel Vaa3D raw files (grayscale only)	folder	unlimited	Uncompressed. Efficient access to subregions. Efficient access to subsets of channels.

[1] If the expected size of a new file is larger than 4 Gbytes, the BigTiff format is used.

1.7 Stitching 2D images

TeraStitcher has been initially designed to process 3D images. It can nevertheless process 2D images too if command line options are set accordingly to the following guidelines.

- In step 2 of TeraStitcher the search space along D dimension must be set to zero (option --sD=0).
- If the n^{th} resolution of the stitched image should be generated (n>0), the flag –isotropic must be added in the command line of TeraConverter, and a value for voxel size *vxl_D* in D dimension (field `D` of tag `voxel_dims` in the xml import file) must be specified that satisfies the relation:

$$vxl\_D \geq n \max(vxl\_V, vxl\_H)$$

where *vxl_V*, and *vxl_H* are the sizes of the voxel in V and D dimensions, respectively (fields `V` and `H` of tag `voxel_dims` in the xml import file).

1.8 Support for multi-channel images

There are two types of multi-channel datasets.

The first one is when all channels of a given tile of the RUI are stored in the same file(s). Allowed formats are:

- RGB TIFFs (both sngle and multi-page) with channels interleaved
- HDF5 formats

It is also assumed that all channels in each file are co-registered.

To perform stitching in this case, one channel has to be chosen to perform both the import and the alignment step. The channel must be specified using letters ‘R’, ‘G’, ‘B’ for the first three channels in the command line option `--imin_channel`. For HDF5 formats, digits can

be used to specify the channel (digits in the range 0-2 may also be used in substitution of letters 'R', 'G', 'B'). To perform the alignment step, the most informative channel should be chosen, i.e. the one containing the structures most suited for computing alignments with the MIP-NCC algorithm (see [1] for details). The resulting xml file should be then processed by steps 3-5 of TeraStitcher. In steps 3-5 it is not required to specify the input channel.

The TeraConveter can then be used to generate a multi-channel stitched image, using the "TIFF (unstitched, 3D)" format as an input format (see section 1.6 *Stitched image generation* for details), and specifying as source the xml file produced by step 5 of TeraStitcher.
By default TeraConverter generates a stitched image containing all channels present in the source image. However, a subset of channels to be converted can be specified using the option `--clist`, that accepts a string of digits, each of which identifies one channel. Only the first 10 channels (using digits from 0 to 9) can be specified with this method.

Note that TeraConverter can generate RGB images only if the source image has up to three channels. If channels are more than three, HDF5 based formats or the "TIFF (tiled, 4D)" format can be used to generate a multi-channel output image. Nevertheless, if the option `--clist` is used to specify that up to three channels must be generated, TeraConverter can be still used to generate RGB images from RUIs with more than three channels.

The second case is when each channel is stored in different file(s), i.e. channels are actually separate datasets. This case is supported by TeraStitcher providing that:

- each channel is stored in file(s) different from the others;
- the RUIs corresponding to each channel have exactly the same structure (i.e. the tile matrices have all the same size in each dimension, tiles have all the same size, overlaps between adjacent tiles are the same for all channels, voxel size is the same for all channels);
- homologous tiles of all channels are co-registered.

To perform stitching in this case, first all channels have to be imported as they were single-channel datasets. Two cases, should be distinguished. The first is when channels are stored not only in different files but also under different root directories (i.e. only files of one channel are under a given directory). In this case, the root directory in the xml import file (attribute `value` of tag `stacks_dir`) of each channel is different and the import can be done as usual. Then you can proceed as explained below. Conversely, if files corresponding to different channels are stored in the same directories, the root directory in the xml import file of each channel is the same and a conflict arises on the auxiliary metadata file `mdata.bin` that is stored in the root directory by default. In this case, an additional tag in the xml import file must be used to specify a different metadata auxiliary file name (and possibly path) for each channel. This can be achieved by specifying the command line option:

`--mdata_bin="`*path and name of the auxiliary metadata file*`"`

when each channel is imported. For instance, if channels 0, 1 and 2 are stored in different files, but all in the same directories under a root `/User/myname/mydataset`, when importing files corresponding to channel 0 the option:

```
--mdata_bin=/User/myname/mydataset/mdata_ch0.bin
```

should be added to the command line.

Using the `-mdata_bin` command line option adds in the generated xml import file after the tag `stacks_dir` the line:

`<mdata_bin value="`*path and name of the auxiliary metadata file*`" />`

With this change, each channel can be imported separately without conflicts (a different metadata auxiliary file is generated for each channel) and the procedure can then proceed as follows. See Appendix C for a complete example.

After all channels have been imported, one channel has to be chosen to perform the alignment step as it were a single-channel dataset. The most informative channel should be chosen, i.e. the one containing the structures most suited for computing alignments with the MIP-NCC algorithm (see [1] for details). The resulting xml file should be then processed by steps 3-5 of TeraStitcher.

After these steps have been completed, we have for the chosen channel an xml file containing the correct positions of tiles in the 3D matrix corresponding to the final image, and for all other channels the xml import file. All these files have to be put in an empty folder. Note that they should be named in such a way that the order with which files are listed does correspond to the desired order of channels (i.e. the first listed xml file is that generated by the dataset corresponding to channel 0, the second listed xml file is that generated by the dataset corresponding to channel 1, and so on). If *path* is the complete path of this folder, a new import step has to be executed where the command line options `--volin`, `--volin_plugin` and `--imin_channel` have the values: *path*, "`MultiVolume`" and the identifier of the channel chosen for alignment computation, respectively. A new xml import file is generated, which has the form reported in the Appendix C. Alternatively, this xml import file can be generated with an additional tool.

The TeraConveter can then be used to generate a multi-channel stitched image, using the "TIFF (unstitched, 3D)" format as an input format (see section 1.6 *Stitched image generation* for details), and specifying as source the xml import file produced according to the procedure just described. Also in this case all considerations about channels of the output image made for the case when channels are stored in the same file(s) apply.

1.9 Processing Imaris IMS datasets

An input plugin named IMS_HDF5 has been added to read image files according to the Imaris IMS format. To use the plugin, the corresponding check box must be checked before generating the project configuration files with Cmake. This requires that an HDF5 static library is available and its path is visible by Cmake. The tests have been carried out using the release 1.8.16 if the HDF5 library (https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.8.16/obtain51816.html).

Currently, to stitch RUI stored in Imaris IMS format an xml import file must be generated. See sections *The standard import step* and *The import step using an externally generated xml import file* for alternative ways to generate this file.

Once an xml import file has been generated it can be used to run the other steps of the stitching process. In particular, for step 2 (displacements computation), the input plugin IMS_HDF5 has to be specified with the command line option –imin_plugin.

For generating a final stitched image still using the Imaris IMS format, the TeraConverter must be used. The following command line options must be used:

Command option	Description
--sfmt="TIFF (unstitched, 3D)"	input image is unstitched: an xml import file with actual tile positions must be provided
--imin_plugin=IMD_HDF5	input image data is stored in Imaris IMS files
--dfmt="HDF5 (Imaris IMS)"	output multi-resolution image format
-s="*complete path of xml import file with tile positions*"	xml import file describing the RUI including tile positions to be used for stitching
-d="*complete path of output file (.ims suffix)*"	file containing the output stitched image
--mdata_fname="*complete path of ims file with image metadata*" or --mdata_fname=null	any Imaris IMS file containing the general metadata characterizing the image to be stitched; alternatively if 'null' is specified, default metadata are stored in the output image
--resolutions=xxxxxxxxx	Resolutions to be generated (optional, default resolution 0 only). See also section *TeraConverter* for more details

Note that the command line option –mdata_fname must always be specified when the Imaris IMS format is specified for the output image.

The option should specify the path of a file containing the image metadata when also the input RUI is in Imaris IMS format. This file is typically one of the Imaris IMS files generated during the acquisition process. TeraConverter extracts from this file all attributes referring to general

characteristics of the image and add them to the output image. Non-general attributes like the image dimensions both physical and as a matrix, image position in the physical acquisition space, etc., if any, are not transferred to the output image, but they are re-computed for the image to be generated and then added.
For the sake of flexibility, it is possible to determine which attributes that must be transferred from the metadata file to the output image by modifying the list of attributed that must NOT be transferred (currently a recompilation of the code is needed).

Alternatively, if the source RUI is in any other format, the option –mdata_fname can be set to 'null' to generate a fully compatible Imaris IMS output file with default metadata.

1.9.1 Stitching multi-resolution and time-series datasets

A source dataset in Imaris IMS format may contain in the same files multiple resolutions and/or multiple time points of a time series.

To enable the reading of a resolution and a time point different from resolution 0 and time point 0, the optional tags 'subimage' has been added to the xml import file. It has the form:

```
<subimage resolution="n" timepoint="m" />
```

where 'n' and 'm' are integer values starting from 0. The fields 'resolution' and 'timepoint' are optional and the default value is 0 for both fields.

All fields in the xml import file have to be set coherently with the specified resolution and time point (including the default ones). In particular:

- the voxel sizes (tag voxel_dims) must have the values corresponding to the specified resolution
- tile displacements (fields ABS_V, ABS_H, and ABS_D of tag Stack) have to be the ones corresponding to the specified resolution and time point.

All steps of TeraStitcher apply to the resolution and time point specified in the xml import file. This implies that in order to stitch an image at a given resolution using the displacements computed at another resolution, a new xml import file has to be created with the right voxel sizes and displacements (they can be easily computed starting from the ratio between the two resolutions). The same apply for the stitched image generation using TeraConverter (see the warning below).

Note that specifying a resolution and/or a time point has effect only if the source file format is a multi-resolution format and/or can store time-series. Currently only the IMS_HDF5 input plugin supports these features. Other plugins ignore the the subimage tag.

WARNING:
When TeraConverter is used to stitch resolution 'n' of the RUI with 'n' greater than 0 (i.e. using an xml import file where the fields 'resolution' is specified and it is different from 0), the resolution 0 of the output image corresponds to resolution 'n' of the RUI, resolution 1 of the output image corresponds to resolution 'n+1' of the RUI, and so on. The metadata of the output file are set accordingly.

1.9.2 Compression with dynamically loaded filters

The HDF5 library has a few internal compressors (referred to as "filters") that can be used to perform compression/decompression of data. It has also a mechanism to dynamically load external compressors that can process data both in read and write operations. Filters must be dynamically libraries located in a folder which path is stored in the environment variable HDF5_PLUGIN_PATH. Filters are uniquely identified by an integer in interval [0,65535]. A list of HDF5 registered filters can be found at https://support.hdfgroup.org/services/contributions.html. When data in an HDF5 file are compressed the library automatically tries to load the appropriate filter from the folder stored in HDF5_PLUGIN_PATH environment variable. Conversely, the filter identifier together with possible filter parameters have to be passed to the library to compress data when a file is written.

TeraConverter supports dynamically loaded filters both in reading and writing operations on files in IMS Imaris format, providing that the HDF5_PLUGIN_PATH environment variable is set to the path where the requested filter is stored. In read operations the filter is loaded automatically to decompress data, whereas when data have to be compressed before write them to disk the filter has to be explicitly specified together with possible filter parameters.

To specify a filter to be used in file generation, the command line option:

`--compress_params=`*ID*`[:`*param1*`:`*param2*`: ...]`

where *ID* is the unique identifier of the filter, and *param1*, *param2*, … are unsigned integers corresponding to the (optional) configuration parameters accepted by the filter.

2 TeraConveter (source code only)

TeraConverter is a command-line tool for **converting** terabytes (and more) of multidimensional (3/4/5D) image data from/to different formats.

Specifically, it can:

- **convert** one format (e.g. TIFF series) to another (e.g. Imaris IMS file format)
- generate a **multiresolution pyramid** suited for **real-time visualization** with TeraFly, BigDataViewer and Andor/Bitplane tools
- **stitch** multiple image stacks given their positions (this replaces the Merge step of TeraStitcher)
- perform any of the previous tasks on images being part of a time series
- perform any of the previous tasks on a x-y-z **subset** of the input data, with one or multiple CPUs (for more details about **parallel execution**, see ParaConverter)

2.1 Command line options

`--sfmt=<string>`
Any of the supported input formats listed in the Supported file formats section.

`--dfmt=<string>`

Any of the supported output formats listed in the Supported file formats section.

`-s=<string>`
Path of source image (it is the path of a folder or a file according to the input format.

`-d=<string>`
Path of destination image (it is the path of a folder or a file, according to the output format.

`--resolutions=<string>`
Resolutions indices (layers l) of the image pyramid to be produced. l must be in [0,10] and 2^l is the corresponding subsampling factor. Default is 0, that means that only the original resolution (subsampling factor 2^0=1) will be produced. For instance: 0123 means pyramid layers 0, 1, 2 and 3 will be produced; 25 means pyramid layers 2 and 5 will be produced; and so on.

`--width=<integer>, --height, --depth`
Tiles dimension along x, y and z, respectively (for tiled output formats, only).

`--H0=<integer>, --H1, --V0, --V1, --D0, --D1`
[H0,H1) x [V0,V1) x [D0,D1) define the x-y-z subset of the input data to process. Note that intervals are open on the right.

`--libtiff_uncompress`
Configure the LibTIFF library to not compress output files (default: compression enabled)

`--libtiff_bigtiff`
Forces the creation of BigTiff files (default: BigTiff disabled).
WARNING: if the expected size of a new file is larger than 4 GBytes, the BigTiff format is used even if this flag has not been set. This rule applies to compressed files too, since it is difficult to predict if compression turns out in an actual file size lesser or equal than 4 GBytes.

`--libtiff_rowsperstrip=<integer>`
Configure the LibTIFF library to pack *n* rows per strip when compression is enabled (default: 1 row per strip). To have just one strip per slice (i.e. to pack all the rows of a slice in a single strip), this option must be set to -1.

`--timeseries`
Apply the conversion to all images stored into the source path (option `-s`), which must be a folder (see also section 2.6)

For other command line options launch TeraConverter with `-h` or `--help` option and/or additional information below.

2.2 Supported file formats

TeraConverter accepts the following formats (in square brackets is specified if the format is supported for both input and output image of for the input image only).

- **TIFF (3D)** [input]

a multipage 3D TIFF file (grayscale or RGB)

- **TIFF (series, 2D)** [input/output]
  a folder containing a series (1+) of 2D TIFF files (grayscale or RGB)

- **TIFF (tiled, 2D)** [input/output]
  y-x-z hierarchy of tiles, each tile is a series of 2D TIFF files (grayscale or RGB)

- **TIFF (tiled, 3D)** [input/output]
  y-x-z hierarchy of tiles, each tile is a series of multipage 3D TIFF files (grayscale or RGB)

- **TIFF (tiled, 4D)** [input/output]
  c-y-x-z hierarchy of tiles, each tile is a series of single-channel multipage 3D TIFF files (grayscale only)

- **Vaa3D raw** [input]
  a 3D Vaa3D raw file (grayscale or multi-channel)

- **Vaa3D raw (series, 2D)** [input/output]
  a folder containing a series (1+) of 2D Vaa3D raw files (grayscale or multi-channel)

- **Vaa3D raw (tiled, 2D)** [input/output]
  y-x-z hierarchy of tiles, each tile is a series of 2D Vaa3D raw files (grayscale or multi-channel)

- **Vaa3D raw (tiled, 3D)** [input/output]
  y-x-z hierarchy of tiles, each tile is a series of 3D Vaa3D raw files (grayscale or multi-channel)

- **Vaa3D raw (tiled, 4D)** [input/output]
  c-y-x-z hierarchy of tiles, each tile is a series of 3D single-channel Vaa3D raw files (grayscale only)

- **HDF5 (BigDataViewer)** [input/output]
  a HDF5 file containing a BigDataViewer 3D image (grayscale or multi-channel)

- **HDF5 (Imaris IMS)** [output]
  a HDF5 file containing a Imaris 3D image (grayscale or multi-channel)

- **TIFF (unstitched, 3D)** [input]
  a TeraStitcher XML descriptor pointing to a valid **unstitched image** (tiles of the unstitched image can be stored in any format accepted by TeraStitcher (see section 1.1)

Formats: TIFF (tiled, 2D), TIFF (tiled, 3D), TIFF (tiled 4D), Vaa3D raw (tiled, 2D), Vaa3D raw (tiled, 3D), and Vaa3D raw (tiled 4D) are *externally tiled formats*, i.e. the output image is partitioned into multiple files, and in each file is stored a contiguous sub-region of the image.

2.3 Tiled 4D formats

Among allowed output formats there are “TIFF (tiled, 4D)” and “Vaa3D raw (tiled, 4D)”. These formats provide a number of advantages for multi-channel datasets:

- They are not limited in the number of channels.
- Channels are represented as separate and independent single channel images with a valid (tiled, 3D) structure. If individual channels have to be manipulated only files belonging to those channels must be accessed.
- New channels can be dynamically added, providing that all channels of the image have the same characteristics (basically image size, color depth, reference system, and voxel size).

To exploit the latter feature, each channel must be converted separately using the same output folder (option `-d=`*output_directory*) and adding the command line option `--ch_dir=`, with a different string for every channel. When all the desired channels have been added, the metadata files have to be regenerated with the command:

```
mdatagenerator -r=output_directory -sfmt=tiled_4D_format --update
```

where *output_directory* is the output directory used to perform conversion. Note that if other channels are added at a later time, the metadata can be regenerated with the same command repeatedly.

2.4 Using TeraConverter for stitched image generation

Since one of the allowed inputs is an xml file specifying the alignments of an unstitched image according to TeraStitcher conventions, TeraConverter can be used in place of the last step of TeraStitcher. In other words, TeraStitcher can be used to import and unstitched image (step 1) and compute the correct tiles alignments (steps: 2-5) and then the resulting xml file can be given as an input to TeraConverter (“TIFF (unstitched, 3D)” input format) that can generate the corresponding stitched image in any of the output formats it supports. This introduces a great flexibility in the generation of the stitched image.

2.5 Lossy compression

When images are very large (much larger than 1 TByte) their manipulation may become unacceptably expensive in term of time or space or both. In this case, resorting to lossy compression can be an option to reduce the cost of simple operations like visualization or copy.

The command line option:

```
--rescale=<integer>
```

sets to 0 the *n* least significant bits of each voxel of the output image, where *n* is the integer specified in the option. This corresponds to map each voxel value to values that are multiple of 2^n. If the image is saved in a compressed format, this mapping may remarkably increase the compression ratio.

If the output formats are tiled (e.g. “TIFF (tiled, xD)” or “HDF5 (xxx)”), compression ratio may by further increased by combining the `--rescale` option adding the command line option:

```
--libtiff_rowsperstrip=-1
```

because compression ration increases as long as more rows are packed in one strip. However, do not use this option if the image is very large in X-Y dimensions and the output format is “TIFF (series, 2D)”, because in this case the reading performance would dramatically drop.

2.6 Time series

If the flag `--timeseries` has been specified, the path of the source must be a folder containing a series of images corresponding to the timepoints of a time series. All images of the time series must have the same dimensions, channels, color depth, etc. Teraconverter applies the conversion to each image in the folder and generates as many output images as are the timepoints.

3 ParaTools

ParaTools are python scripts that use an implementation of MPI (e.g. OpenMPI, MPICH, proprietary implementations, etc.) and the package mpi4py (http://pythonhosted.org/mpi4py/) to execute multiple instances of the TeraTools in parallel. See Appendix D for details about the tools to be installed and other information on how to use ParaTools.

ParaTools have been designed to parallelize the most computation intensive steps of the stitching pipeline, i.e. step 2 (alignment computation) and step 6 (tile fusion and generation of the final stitched image). To avoid confusion we clarify that in other documents we have used the term *align step* to refer to step 2 and *fusion step* to refer to step 6.

3.1 Parastitcher

Parastitcher executes a pipeline of commands that launches multiple instances of TeraStitcher in parallel performing each the displacement computation (step 2) on a portion of the dataset. The script includes an algorithm for efficient partition of the image to be converted and the needed synchronization between the serial and the parallel part of the pipeline.

Parastitcher can be launched in several ways, depending on the parallel platform on which it is executed. On high-end workstations that have no special scheduler for controlling job execution, Parastitcher must be launched using *mpirun* or equivalent commands. In this introduction we will use in all example *mpirun*. The command to be issued must have the form:

```
mpirun -np num_procs python
      parastitcherX.Y.Z.py -2 --projin=import_xml_file
                  --projout=displacement_xml_file    other_options
```

where:

- *num_procs* is the desired level of parallelism plus one, i.e. how many MPI processes are launched (one of the launched process is the master and does not perform any useful work);
- in the script name `X` is the major version, `Y` is the minor version, `Z` is the patch;
- *import_xml_file* is the complete path of the import xml file generated following one of the approaches discussed in section 1.2;
- *displacement_xml_file* is the complete path of the xml file containing the results of the displacement computation on the whole dataset;
- *other_options* are any other the command line options that should be passed to TeraStitcher.

Parallel execution of step 2 with Parastitcher produces an xml file with all computed alignments (the same it had been generated by a sequential execution of step 2), which can be given as an input to step 3 of TeraStitcher.

Since versione 3 of Parastitcher, the script can be used to perform also step 6 of the stitching pipeline. In this case the script launches multiple instances of TeraConverter and transparently maps the command line options of TeraStitcher on those required by

TeraConveter so as the users deal with a unique command line interface (the one of TeraStitcher).
To perform step 6 with Parastitcher3 The command to be issued must have the form:

```
mpirun -np num_procs python
      parastitcher3.Y.Z.py -6 --projin=import_xml_file
                  --volout=fused_image_destination  other_options
```

where:

- *num_procs* is the desired level of parallelism plus one, i.e. how many MPI processes are launched (one of the launched process is the master and does not perform any useful work);
- in the script name `X` is the major version, `Y` is the minor version, `Z` is the patch;
- *import_xml_file* is the complete path of the import xml file generated following one of the approaches discussed in section 1.2;
- *fused_image_destination* is the complete path of the folder where the fused image has to be saved;
- *other_options* are any other the command line options that should be passed to perform the fusion.

The *other_options* should be passed in the same form as it is required by TeraStitcher. In particular:

- `--volout_plugin` specifies the format to be used for the generation of the fused image; possible values are: "`TiledXY|2Dseries`" (default value) to generate a tiled image in which tiles are series of 2D TIFFs, and "`TiledXY|3Dseries`" to generate a tiled image in which tiles are series of multipage TIFFs;
- `--sliceheight`, `--slicewidth`, `--slicedepth` specify the size of the tiles to be generated in V, H, and D direction, respectively; for all options the default is 256.

It is worth noting that using Parastitcher3 to generate the final fused image has some limitations with respect to using Paraconverter (see section 3.2). In particular, Paraconverter allows more formats for the fused image.

3.1.1 Hints for performing step 2 with Parastitcher

1. One parameter that can be used to control how the dataset is partitioned for parallel execution is the value of the command line option `--subvoldim`. , which default value is 200. More specifically, let be *D* the total number of slices and *subvoldim* the value of `subvoldim` option. The dataset is initially partitioned in ceil(*D*/*subvoldim*) subregions. If this number can saturate the requested degree of parallelism, i.e. ceil(*D*/*subvoldim*) > 2 × (*num_procs* -1), as many instances of Terastitcher are launched and scheduled on a FIFO bases onto *num_procs* -1 MPI processes, otherwise the dataset is further partitioned at tile level. This further partitioning, however, introduces some I/O overhead and it should be avoided properly setting *subvoldim* if *D* is large enough.
   Just to make an example if *D* = 3430 and the requested degree of parallelism is 10, *subvoldim* should be set to a value not larger than =180 to avoid partitioning at tile level. Consider however, that the value of *subvoldim* should not be set to an artificially low value, since the amount of computation needed for alignment grows roughly inversely proportionally with *subvoldim*.

2. Running multiple instances of TeraStitcher in parallel might require much more memory than running a single instance with the same options. To have an rough estimate of memory requirements the following formula can be used:

   $$slice\_height \times slice\_width \times 4 \times (\min(n\_tiles\_V, n\_tiles\_H) + 1) \times subvoldim \times (num\_procs - 1)$$

   where:
   - *slice_height* is the height of a slice of a tile;
   - *slice_width* is the width of a slice of a tile:
   - *n_tiles_V* is the number of rows of the tile matrix
   - *n_tiles_H* is the number of columns of the tile matrix

3.2 Paraconverter

Paraconverter executes a pipeline of commands that launches multiple instances of TeraConverter in parallel. The script includes an algorithm for efficient partition of the image to be converted and the needed synchronization between the serial and the parallel part of the pipeline.

Paraconverter can be launched in several ways, depending on the parallel platform on which is executed. On highend workstations that have no special scheduler for controlling job execution, Paraconverter must be launched using *mpirun* or equivalent commands. In this introduction we will use in all example *mpirun*. Hence in all cases the command to be issued must have the form:

```
mpirun -np num_procs python
      paraconverterX.Y.Z.py -s=source_volume -d=destination_path
            --depth=dd --height= hh --width= ww --sfmt=source_format
            --dfmt=destination_format --resolutions=rr   other_options
```

where:

- *num_procs* is the desired level of parallelism plus one, i.e. how many MPI processes are launched (one of the launched process is the master and does not perform any useful work);
- in the script name `X` is the major version, `Y` is the minor version, `Z` is the patch;
- *dd*, *hh*, *ww* are the values used to partition the image for parallel execution;
- *source_volume* is the complete path of the directory/file where the image to be converted is stored;
- *destination_path* is the complete path of the directory where the converted image has to be stored;
- *source_format* is one of the following input formats allowed by TeraConverter: “TIFF (3D)”, “TIFF (series, 2D)” , “TIFF (tiled, 2D)” , “TIFF (tiled, 3D)” , “TIFF (tiled, 4D)”, “Vaa3D raw”, “Vaa3D raw (series, 2D)”, “Vaa3D raw (tiled, 2D)”, “Vaa3D raw (tiled, 3D”, “Vaa3D raw (tiled, 4D)”, “TIFF (unstitched, 3D)”;
- *destination_format* is one of the following input formats allowed by TeraConverter: “TIFF (series, 2D)” , “TIFF (tiled, 2D)” , “TIFF (tiled, 3D)” , “TIFF (tiled, 4D)”, “Vaa3D raw (series, 2D)”, “Vaa3D raw (tiled, 2D)”, “Vaa3D raw (tiled, 3D”, “Vaa3D raw (tiled, 4D)”;
- *rr* are the requested resolutions (according to the convention used by teraconverter);

- *other_options* are any other the command line options that should be passed to TeraConverter, excluding option `--V0`, `--V1`, `--H0`, `--H1`, `--D0`, `--D1`, that cannot be used with Paraconverter (in other words Paraconverter cam be used to convert the whole image only).

Note that options `--depth`, `--height`, and `--width` are mandatory when Paraconverter is used since they are used by the partitioning algorithm (see below). See also the comments at the beginning of the script for more details and options. Note also that not all input/output formats allowed by TeraConverter are allowed with Paraconveter.

Paraconverter parallelizes execution according to the master-slave approach. The command launches one master process and *num_procs*–1 slave processes. The master process creates the directory hierarchy where the converted image has to be stored, and then partitions the image to be converted into cuboid subregions and assigns each subregion to a slave process, which runs a different instance of TeraConverter for each subregion to be converted. If the number of subregions is larger than the number of slaves, Paraconverter assings the first *num_procs*–1 subregions to the slaves, and then assigns the remaining subregions to slaves as soon as an instance of TeraConverter terminates its task, until all subregions have been converted. When all subregions have been converted the master generates the metadata needed to manage the converted image, if any.

It must be noted that the specified tile sizes (*dd*, *hh*, *ww*) play the role of upper bounds to the actual tile sizes of the output image. Indeed, the specified sizes are used by the partitioning algorithm to compute an optimal partition of the source image. Actual tile sizes of the output image are guaranteed to be not larger than those specified by the above options, and reasonably near to them.

3.2.1 Hints for using Paraconverter

1. Running multiple instances of TeraConverter in parallel might require much more memory than running a single instance with the same options. To have an estimate of memory requirements the script should be launched with all the parameters that have to be used for the conversion and adding the option `--info`. In this case the conversion does not take place, but information about the partition that will be used is sent to standard output, including an estimate memory occupancy in Gbytes in the form:

   `Memory needed for` *num_procs-1* `concurrent processes:` *N* `GBytes`

   Memory occupancy depends on many factors, but as a rule of thumb, it can be assumed that it increases proportionally to the requested parallelism degree (*num_procs-1*) for datasets with a number of slices greater than:

   $$2 \times (num\_procs\text{-}1) \times dd$$

   where *dd* is the value of option `--depth` above, while the increase is less than proportional if the number of slices smaller.

2. Image conversion is a highly I/O bound process. Excluding high performance machines with a highly parallel file system where I/O throughput scales with the number of processing elements used, usually I/O operations represent a fraction of the whole workload that cannot be parallelized. Hence, parallelism can be effectively exploited as

long as the throughput of the I/O system is not saturated. For the Amdahl law, when this happen, increasing the degree of parallelism does not reduce further the conversion time.

3. Since only externally tiled formats are allowed for the output image, the user should choose between TIFF-based and Vaa3D raw-based formats.
   TIFF-based formats can be compressed (is the default) which means that conversion requires more computation, but the output image is smaller (a compression factor of 4 has been observed for images with large empty regions) and more parallelism can be exploited (10x speedups have been observed with SATA disks). Conversely, Vaa3D raw-based formats require more space on disks and a limited degree of parallelism can be effectively exploited (2-3x speedups have been observed on the same I/O subsystem). When all exploitable parallelism is used, the total completion time of the conversion of the two output formats is comparable.
   These considerations suggest that TIFF-based format could be preferable, especially if images have large empty sub-regions.

4. For images with thousand of planes in Z, tiled 2D output formats should be avoided to limit the number of files generated by the conversion.

5. For images with more than three channels, the format "TIFF (tiled 3D)" cannot be used, unless channels are converted separately. In this case, the format "TIFF (tiled, 4D)" can be used if a TIFF-based format is preferred. This can be done in two ways:
   - all channels are converted together simply specifying "TIFF (tiled, 4D)" as output format
   - one channel at the time is converted, and the image generated by each conversion is incrementally added in order to have a complete TIFF (tiled, 4D) image at the end of the process; to follow this alternative the option `--ch_dir=`*channel_folder_name* must be specified at each conversion, using a different name for every channel and specifying for all channels the same output folder (option `-d=`*output_directory*); when all desired channels have been added, the metadata have to regenerated (see TeraConverter documentation for more details).

4 Importing 3D tiled RUIs (*source code only*)

TeraStitcher has been initially designed to stitch RUIs consisting of overlapping tiles arranged according to a 2D regular matrix along X-Y dimensions. Nevertheless, there are acquisition systems that generate RUIs consisting of overlapping tiles arranged according to a 3D regular matrix.

A new experimental tool has been developed to deal with stitching these 3D tiled RUIs referred to in the following as TeraStitcher3D. The tool is still under development, but it is possible already execute the whole pipeline needed to stitch a 3D tiled RUIs using a command line tool named TeraStitcher2 (not documented yet).

A critical step of the TeraStitcher3D stitching pipeline is the import step, which relies on the ability to use externally generated import xml files to cope with the much more complex structure of 3D tiled RUI.

Before providing further details on how to make available to the tool the information it needs to import the RUI, we introduce some terminology and conventions.

4.1 Terminology and conventions

A 3D image *stack* is a series of equally sized 2D images (*planes*) that form a 3D matrix of voxels. Stacks are assumed to be partially overlapped tiles of a larger 3D image, arranged according to a regular grid. The tile grid can be either bi-dimensional (i.e. tiles are arranged according to a 2D matrix and their positions are identified by two indices), or tri-dimensional (i.e. tiles are arranged according to a 3D matrix and their positions are identified by three indices).

As explained in section *The reference system*, we assume that the three dimensions of a stack are referred to as *vertical* (V), *horizontal* (H), and *depth* (D), where V is the vertical dimension of a plane viewed as a matrix, H is the horizontal dimension of a plane viewed as a matrix, and D is the dimension perpendicular to the planes. Note that this convention is related to the actual orientation of the image planes and not to the physical coordinates of the acquisition system.

A stack (or tile) may physically consist of:
a) a series of files storing one 2D image each
b) multiple files storing one 3D substack of the stack each
c) a single file storing the whole 3D stack.

In case the tile grid is a 3D matrix, the set of all tiles corresponding to a given position along D is referred to as a *layer*. Note that a layer is a 2D matrix of partially overlapped tiles.

4.2 Writing a .ini file describing the RUI configuration

A tiled RUI (both 2D and 3D) can be imported using the python scripts mentioned in section *The import step using an externally generated xml import file* plus the python script **terastitcher3D.py** which is a driver that executes the stitching pipeline automatically.

All these scripts read the RUI metadata from a text file written according to the following rules:

- lines starting with the character '#' are comments
- lines that are not comments contain a *key* or a *section header*
- a key (or a property) consists of a name (case sensitive) and a value, delimited by the symbol '=', e.g. keyname = value; a key can also have multiple values, separated by a space or a tab, e.g. keyname = value1 value2 ...
- a section header is a name (case sensitive) delimited by square brakets and preceeds a group of keys associated with that section.

The following sections and keys are mandatory:

- section [format] has keys:
  - *tiling*: can assume values '2D' or '3D'; '2D' means that tiles are organized in a 2D matrix, '3D' means that tiles are organized in a 3D matrix
  - *filetype*: can assume values 'slice', 'block' or 'stack'; 'slice' means that stacks are stored as series of 2D image files, 'block' means that stacks are stored as one or more 3D image files, 'stack' means that stacks are stored as one single 3D image file (the case when multiple stacks are stored in a same single file is included in this case)
  - *sparse*: must assume the fixed value 'false' (for future use)
- section [reference system] has keys:
  - *vertical*: can assume values 'X', 'Y', or 'Z', possibly preceded by the symbol '-' (minus); specifies which element of the triplets specifying spatial properties (X, Y or Z) maps to dimension V; if the symbol '-' is not present, spatial coordinates (see *origin* and *spacing* keys description below) increase with tile matrix indices (see *stack* key description below), otherwise spatial coordinates decrease with tile matrix indices
  - *horizontal*: can assume values 'X', 'Y', or 'Z', possibly preceded by the symbol '-' (minus); specifies which element of the triplets specifying spatial properties map to dimension H; if the symbol '-' is not present, spatial coordinates (see *origin* and *spacing* keys description below) increase with tile matrix indices (see *stack* key description below), otherwise spatial coordinates decrease with tile matrix indices
  - *depth*: can assume values 'X', 'Y', or 'Z', possibly preceded by the symbol '-' (minus); specifies which element of the triplets specifying spatial properties map to dimension D; if the symbol '-' is not present, spatial coordinates (see *origin* and *spacing* keys description below) increase with tile matrix indices (see *stack* key description below), otherwise spatial coordinates decrease with tile matrix indices
- section [acquisition data] has keys:
  - *origin*: three real numbers specifying the spatial coordinates in mm of a well defined voxel (e.g. the up, left, top corner of the the image cuboid) in the physical space of the acquisition system; these coordinates can be set to (0, 0, 0) if they are not relevant
  - *voxel*: three real numbers specifying the voxel size in um
  - spacing: three real numbers specifying the offsets in um between adjacent tiles in the tile matrix; if the tile matrix is 2D the offset of the dimension associated to D (i.e. the one specified by attribute 'ref_sys') must be 0
  - *channels*: number of channels
  - *colordepth*: number of bits per channel
- section [grid] has keys:
  - *rootdir*: root directory where RUI is stored

- *dims*: three real numbers specifying the dimensions of the tile matrix (how many tiles in each dimension); if it is a 2D matrix the value of the dimension associated do D (i.e. the one specified by attribute 'ref_sys') must be 1
- *stack*: this is a muti-value key providing all the needed information concerning a single tile; the first value is the relative path (with respect to the value of the key *rootdir* above) of the directory where all data belonging to the tile is stored; the second, third and fourth values are the indices X, Y and Z of the tile in the tile matrix; the fifth value is a regular expression identifying all and only the names of the file in the directory of the tile that contain planes of that tile; the sixth value is used only if more tiles are stored in the same physical file (e.g. in the .czi file format) and it is used to identify the sub-image in the file containing the data of that tile; in the .ini file there must be as many stack keys as are the tiles in the tile matrix (i.e. the result of the product of the three values of the key *dims*).

All triplets related to a spatial property (e.g. *origin*, *voxel*, *spacing*, *dims*, *stack indices*) refer to dimensions X, Y, Z, respectively. Their mapping to V, H, D is specified by keys *vertical*, *horizontal*, and *depth*.

4.3 Constraints on RUI configuration

The procedure designed to import 3D tiled RUIs makes a few assumptions that must be satisfied by the acquisition system. They are:

- All tiles in the same layer must have the same number of planes
- Data (files) corresponding to one tile must be all stored in the same folder; if data corresponding to different tiles are stored in different files in the same folder, the names of the files corresponding to a given tile should be distinguishable by others files in that folder using a regular expression
- In each dimension, spacing among tiles imposed by the acquisition system (i.e. nominal offsets between adjacent tiles) must be the all same

5. Stitching 3D tiled RUIs (*source code only*)

An experimental tool names TeraStitcher2 is under development to enable stitching of datasets organized as a 3D matrix of tiles. The tool has been designed to deal with datasets generated by acquisition systems that scan a layer at the time of the sample, typically by physically slicing the sample.

<*work in progress: documentation to developed*>

Appendix

A. Using CUDA implementation of MIP-NCC alignment algorithm

From version 1.11 of TeraStitcher, a CUDA implementation of the MIP-NCC alignment algorithm has been included in the distribution. In order to execute the MIP-NCC alignment algorithm on NVIDIA accelerators set the environment variable:

USECUDA_X_NCC=1

Use nvprof to check if the GPU is actually used by TeraStitcher.

Before enabling the execution of CUDA code, a check should be done about compatibility of the GPU device and its driver with the Toolkit used to generate the TeraStitcher binaries. TeraStitcher has been generated using NVIDIA CUDA Toolkit 7.5 and Visual Studio 2013 Community.

In order to check GPU device and driver compatibility, right-click on desktop, select *NVIDIA configuration panel*. After the panel opens, go to *help->system information*. In the *Display* tab you find which is the GPU device and its driver. Check if they are compatible with NVIDIA CUDA Toolkit 7.5. If the device is compatible, but the driver not, download and install the latest driver for your device at http://www.nvidia.it/Download/index.aspx?lang=en.

B. Requirements for running the LQP global optimization algorithm

To run the LQP algorithm, a python 2 interpreter must be available and standard packages *numpy* (http://www.numpy.org) and *scipy* (https://www.scipy.org) installed.

Moreover, the python script `LQP_HE.py` (which is separately provided) must be placed somewhere and the environment variable `__LQP_PATH__` must be set to the path where the script has been placed.

For instance, under **Linux** or **Mac OS X**, assuming that the script is placed in folder:

```
/Users/johndoe/mypyhthonscripts
```

to run LQP algorithm on file xml_displthres.xml, containing the computed alignments between tiles with their reliabilities, the commands should be issued:

```
export __LQP_PATH__=/Users/johndoe/mypyhthonscripts
terastitcher -5 --projin=xml_displthres.xml --algorithm=LQP
```

Under **Windows**, assuming that the script is placed in folder:

```
C:\Users\johndoe\mypyhthonscripts
```

to run LQP algorithm issue the commands:

```
set __LQP_PATH__=C:\Users\johndoe\mypyhthonscripts
terastitcher -5 --projin= xml_displthres.xml --algorithm=LQP
```

C. *Xml import file for datasets storing channels in separate files*

To process multi-channel RUIs in which channels are stored in separate files, first all channels must be imported separately.

If files of different channels are under the same root directory, the xml import files of all channels must use the `mdata_bin` tag according to the following example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TeraStitcher SYSTEM "TeraStitcher.DTD">
<TeraStitcher volume_format="TiledXY|2Dseries" input_plugin="tiff2D">
    <stacks_dir value="path_of folder_where_the_xml_files_of_all_channels_are_stored" />
    <mdata_bin value="path_and_name_of_the_ metadata_auxiliary_file" />
    <voxel_dims V="V_size" H="H_size " D="D_size " />
    <origin V="V_orig_coord" H="H_orig_coord " D="D_orig_coord " />
    <mechanical_displacements V="V_mech_displ" H=" H_mech_displ " />
    <dimensions stack_rows="#rows" stack_columns="#cols" stack_slices="#slices" />
    <STACKS>

        … attributes and displacements of stacks …

    </STACKS>
</TeraStitcher>
```

After import, one of them should be used to compute the correct tile positions (see section 1.8 for details), then a xml file with the following structure must be used to perform the merge step and generate a multi-channel stitched image.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TeraStitcher SYSTEM "TeraStitcher.DTD">
<TeraStitcher volume_format="MultiVolume" input_plugin="MultiVolume">
    <subvolumes_dir value="path_of folder_where_the_xml_files_of_all_channels_are_stored" />
    <ref_sys ref1="axis_V" ref2=" axis_H" ref3=" axis_D" />
    <voxel_dims V="voxel_size_V" H=" voxel_size_H " D=" voxel_size_D" />
    <origin V="origin_V" H=" origin_H" D=" origin_D" />
    <mechanical_displacements V="-1" H="-1" />
    <dimensions stack_rows="-1" stack_columns="-1" stack_slices="-1" />
    <SUBVOLUMES N_SUBVOLUMES="#channels" ENABLED_SUBVOLUME="main_channel" ALIGNED="false">
        <Subvolume xml_fname="path_and_name_of_xml_file_of_main_channel" />
        <Subvolume xml_fname="path_and_name_of_xml_file_of_other_channel" />
                . . .
    </SUBVOLUMES>
</TeraStitcher>
```

where *main_channel* (value of attribute `ENABLED_SUBVOLUME`) is the digit identifying the channel chosen to compute the correct tile positions that are applied to al channels.

This xml file can be generated by an external tool or by TeraStitcher following the procedure described in section 1.8. Note that:

- In tag `mechanical_displacemets` the values of attributes `V` and `H` are not used and they should be set to -1;
- In tag `dimensions` the values of attributes `stack_rows`, `stack_columns` and `stack_slices` are not used and they should be set to -1;
- In tag `SUBVOLUMES`: attribute `N_SUBVOLUMES` is the numer of channels, attribute `ENABLED_SUBVOLUME` is the identifier of the channel whose tile positions have

been computed and are used for all other channels, and `ALIGNE` is not currently used and it should be set to `false`.

D. Using ParaTools

To use Parastitcher and Paraconverter on the following tools have to be installed:

- MPI;
- Python 2;
- mpi4py package installed (install MPI first).
- The following executables have to be installed :
  - terastitcher (at least version 1.10)
  - mergedisplacements
  - teraconverter
  - mdatagenerator

On Linux and MacOS platforms Open MPI has been tested.

On Windows platforms Microsoft MPI (version 8.1) has been tested. Since Microsoft MPI uses *mpiexec* instead of *mpirun* to launch multiple MPI processes, substitute *mpiexec* to *mpirun* in examples given in section 3.

All Python scripts can be easily adapted to the actual platform where they are executed and to specific needs . Here we provide some additional information useful in this respect.

1. The scripts needs to know where the executables of TeraStitcher and TeraConverter are located. For this reason at the beginning of the script there is a global variable named `prefix` that control where the executables are searched. If `prefix` is set to the empty string the folder where the executables are stored needs to be in the PATH environment variable. Conversely, `prefix` must be set to the complete path of the foder where the executable are stores ('./' if executables are in the current folder).
2. The scripts generate a rich output including all the commands issued to launch instances of TeraStitcher or TeraConverter. It is convenient redirect this output to text file for later verification. In particular the output reports detailed information about execution times of single commands and of whole execution.
3. Script Paraconverter3.X.X.py has the global variable `debug_level` to enable additional output information. In particular, it saves in text files named `output_XX.out`, and located in the current folder, the output of all commands issued.
4. Script Paraconverter3.X.X.py has a `PARAMETERS` section where the values of parameters controlling some default values can be changed according to needs.

E. Using the Bioformats2D and Bioformats3D plugins

To use the plugins relying on the Bio-Formats library the following tools have to be installed:

- Java Runtime Ennvironment
- Java Development Kit
- Bio-Formats library (file ***bioformats_package.jar***)

Both plugins are *input* plugins, i.e. they can be used to read images, but not to write them.

The plugins has been tested on Windows, Linux and Mac OS X with release 5.0.3 of the Bio-Formats library (download at: https://downloads.openmicroscopy.org/bio-formats/5.0.3/). Depending of the Java version installed in your machine, latest versions of the Bio-Formats library may also be used.

The environment variable `__BIOFORMATS_PATH__` must be set to the complete path <*bioformats_path*> where the `bioformats-package.jar` file is stored.

On Linux and Mac OS X:

`export __BIOFORMATS_PATH__=`<*bioformats_path*>

On Windows:

`set __BIOFORMATS_PATH__=`<*bioformats_path*>

N.B. <*bioformats_path*> must specify only the path where file `bioformats-package.jar` is stored. It should not contain the file name of the .jar file.

F. *Stitching multi-channel images when channels in the RUI are stored in separate files.* (**This procedure is deprecated: a much simpler procedure is described in section 1.8 and Appendix C**)

There are two cases:

a) A separate stitched monochromatic image has to be generated for each channel, with the constraint that co-registration between homologous tiles must be maintained after stitching (implying that the stitched images of all channels have the same size in all dimensions).
b) All, or a subset, of channels have to be stored in a single multi-channel image (e.g. using RGB TIFF, HDF5 formats, etc.).

Case a)
Perform the following steps:

1. Chose the most suited channel for alignment computation (see section 1.8).
2. Execute steps 1-5 on this channel (using option `--imin_channel` in steps 1 amd 2). This can be done also using the GUI of TeraStitcher (steps “Import” and “Align”). Assuming without loss of generality that this channel is channel 0, let `xml_merging.xml` be the file so generated. For the sake of clarity rename it as `xml_merging_CH0.xml`.
3. Execute step 1 on all channels (or generate in any way the corresponding xml import file, see section 1.2). For the sake of clarity, let these files have names `xml_import_CH1.xml`, `xml_import_CH2.xml`, etc.

4. From each file obtained at step 3 generate the xml files `xml_merging_CH1.xml`, `xml_merging_CH2.xml`, etc., that are identical, except for attributes `ABS_V`, `ABS_H`, and `ABS_D` that must have the same values of `xml_merging_CH0.xml` in correspondence of homologous tiles (i.e. in `Stack` tags with the same values of `ROW`, `COL` attributes).
5. For every channel, run the command line version of TeraConverter (see section 2):
   - using as input format "TIFF (unstitched, 3D)" (command line option `--sfmt`);
   - specifying as source the path of the xml files `xml_merging_CH0.xml`, `xml_merging_CH1.xml`, `xml_merging_CH2.xml`, etc. (command line option `-s`);
   - specifying the path of a different directory or file name as a destination (command line option `-d`).

   WARNING: before processing any channel with TeraConverter, the `mdata.bin` file in the directories specified at tag `stacks_dir` in the above mentioned xml files must be deleted.

Case b)
Perform the following steps:

1. Chose the most suited channel for alignment computation (see section 1.8).
2. Execute steps 1-5 on this channel (using option `--imin_channel` in steps 1 amd 2). This can be done also using the GUI of TeraStitcher (steps "Import" and "Align"). Assuming without loss of generality that this channel is channel 0, let `xml_merging.xml` be the file so generated. For the sake of clarity rename it as `xml_merging_CH0.xml`.
3. Execute step 1 on all channels (or generate in any way the corresponding xml import file, see section 1.2). For the sake of clarity, let these files have names `xml_import_CH1.xml`, `xml_import_CH2.xml`, etc.
4. From each file obtained at step 3 generate the xml files `xml_merging_CH1.xml`, `xml_merging_CH2.xml`, etc., that are identical, except for attributes `ABS_V`, `ABS_H`, and `ABS_D` that must have the same values of `xml_merging_CH0.xml` in correspondence of homologous tiles (i.e. in `Stack` tags with the same values of `ROW`, `COL` attributes).
5. For every channel run the command line version of TeraConverter (see section 2):
   - using as input format "TIFF (unstitched, 3D)" (command line option `--sfmt`);
   - specifying as source the path of the xml files `xml_merging_CH0.xml`, `xml_merging_CH1.xml`, `xml_merging_CH2.xml`, etc. (command line option `-s`);
   - using as output format "TIFF (tiled, 4D)";
   - specifying always the same directory as a destination (command line option `-d`);
   - adding the command line option `--ch_dir` with a different string for each channel (it is a good practice to use the same convention used for the names of the xml files, for instance something like `--ch_dir=CH0`). See also section 2.3 for further details on this procedure.

WARNING: before processing any channel with TeraConverter, the `mdata.bin` file in the directories specified at tag `stacks_dir` in the above mentioned xml files must be deleted.

6. After all channels have been processed, run the command:

   ```
   mdatagenerator –r=output_directory –sfmt="TIFF (tiled, 4D)" --update
   ```

   where *output_directory* is the same directory specified as a destination at step 4.

7. Run again the command line version of TeraConverter:
   - using as input format "TIFF (tiled, 4D)";
   - specifying as a source the same directory specified as a destination at step 4.

   All other options can be normally used to control the generation of the final multi-channel image (format, resolutions, tiling, etc.).

Note that in case b) the whole dataset has to be read and written one more time. This can be very time consuming for large datasets, but it is the only way currently available to perform this task with TeraTools.

As a final comment we note that if the constraint that channels must be stitched without preserving the co-registration between homologous tiles can be released, the stitching problem becomes trivial since each channel can be processed independently as usual. Nevertheless, the WARNING highlighted above still holds: if the directory specified at tag `stacks_dir` in the xml files is the same for all channels, before processing any channel, file mdata.bin present in that directory must be deleted.