Amit Mukherjee

Mitralab, CSHL

9 October 2013

Usage Instructions for StackAlign and associated software tools

The *StackAlign* software creates a volumetric stack by aligning a series of 2D section images. A rigid transformation model is estimated for each image in the series, which is then applied to the image for aligning it with the volumetric stack of arbitrary resolution. The package is composed of the following modules:

StackAlign – This is the first module that creates an initial estimate of the transformation. The input to this module is text file containing a list of images. The output of the module contains two text files, and image thumbnails in a specified directory. Additionally, the package also outputs one folder named "DIFF" for inspection and diagnostics[1].

StackRepair – This module provides tools for fixing some of the errors (misalignments) in the stack. It reads the content of the directory created by *StackAlign* (or previous runs of *StackRepair*) and outputs another directory containing similar information. Additionally, the package also outputs one folder named "DIFF" for inspection and diagnostics[2].

ExecuteBatchMultiple - This module runs in Black-n-blue development nodes, and manages job submission to the cluster. The input to this module is one of the text files created by *StackAlign* or *StackRepair.* The module also handles parallelization of job submission and keeps an account of the outputs from the cluster nodes ensuring that no job is lost in the compute nodes.

ApplyTForm16bit / *ApplyTForm8bit* – Applies the transformation to high resolution JP2 images on the compute nodes of Black-n-blue cluster. It splits the image into tiles, applies

[1] This is a new addition.
[2] This is a new addition.

transform to each tile and rejoins the image. This ensures that large images can be handled without requiring enough memory to hold multiple copies of the full image in memory. It applies the transform estimated by uses B-spline interpolation. It also creates the jpg thumbnails necessary for posting the images to the portal.

Following are the highlights of the software

1. The user interaction (input/output) with the software is designed using text files and shell commands. The software does not directly interact with the storage database at any stage[3]. The present implementation is for Linux shell, but can easily be ported to Windows or Mac (if requested).
2. The input to *StackAlign* is a text file containing the full path of images in tif, png or jpg format, such that one image is in one line of the text file.
    a. Any additional information or comment should be preceded by a "#" symbol in the 0^{th} position of the line.
    b. Note that the order of the sequence is determined by the order in which the file names are arranged in the text file. A sequence number is assigned to each image which is the position of the image in the list (i.e. the line number minus the number of comment lines)
    c. One can assign a pre-rotation (about the center of the image). To do this you have to append the filename with a ":" followed by rotation in degrees.

       For example, `/home/amit/StackAlign/JGANissls/JGA1-N88-2013.04.08-20.00.30_JGA1_1_(0)0262.tif:-90` implies that the image would be pre-rotated by an amount 90 degrees in clockwise direction before

[3] However, I am providing with a utility to generate this list automatically from StorageDB. This is only valid for PMD, Hua, MD and PTM brains. The utility `./CreateListFromDatabase [brainname]` scans the database, and automatically creates the list of sections in the proper order, which can be used as input to StackAlign.

input to the registration. This option should only be used if the images are scanned randomly in multiple orientations. Note that if you just provide the file name, it would imply that pre-rotation is 0.

3. *StackRepair* works on the output of *StackAlign.* The software package attempts to create stacks robustly from any set of images. Occasionally, these images may contain tissue / imaging damages or bad cropping. In this sequential chain, the weakest link in the chain determines the quality of alignment, therefore even if there is one bad link in the chain, all the sections succeeding that section would appear misaligned. *StackRepair* provides a tools to repair or correct all broken links. Correction can be in the form of
    a. deleting a bad image and recomposing the stack without the bad image
    b. reattempting the transformation between a pair of images, hoping that stricter convergence criterion would repair the broken link. Additionally, there are optional tools for pre-rotating and border-cropping the sections, which greatly increases the chances of successful alignment. Details are presented later.
    c. Apply a global rotation or translation to the entire stack.

The input and output directories of *StackRepair* are very similar except for the correction/repair that is applied. One purpose of doing a *StackRepair* instead of redoing *StackAlign* (with the bad image removed) is the time consideration, because the good part of the result of *StackAlign* (which is usually more than 90% pair-wise transforms) is not reestimated, hence re-used from previous runs.

4. Both *StackAlign* and *StackRepair* creates a folder by the name “DIFF” inside the output directories. This folder contains difference images between consecutive sections. These difference images can be used to inspect the misalignments quickly. The naming convention of the difference image uses the sequence numbers of the two images.

5. The usage syntax for *StackAlign*, *StackRepair*, *ExecuteBatchMultiple,* etc can be obtained by typing the module name without arguments. Description of all arguments is presented here.
    a. For *StackAlign* the arguments are

```
[amit@mitragpu2 build]$ ./StackAlign

Usage : ./StackAlign List_file [ options ]
Options:
         -brain (required)
         -label (required)
         -output ( default: ./Output )
         -beta_inv (range: 5-100, default: 25)
         -gx (ex: 900, default: 1000)
          -gy (ex: 750, default: 1000)
```

`List_file` is the name of the input text file containing file names. It is a mandatory argument. Other mandatory arguments are `-brain` and `-label`. Note that since this software does not interact with the database directly, the `-brain` can technically be any string of alpha-numeric characters which will be used to name the parameter files. `-label` on the other hand can be “F”, “N” or “IHC”. Note that the bit depth of the images are determined using the `-label` parameter. So if a 16 bit image is processed, it should be labeled as F. Later versions of the software may require user to input bit-depth information instead of inferring the bit-depth from label (but nothing is perfect right!!!).

`-output` is the directory path where all output files and thumbnails will be created. `StackAlign` outputs two text files (containing parameters), transformed thumbnails for all the valid images in the `List_file` and one folder containing DIFF images. All these files will be placed in the output directory. The two output text files have the form `brainName_labelName.txt` and `brainName_labelName_XForm.txt`, where `brainName` and `labelName` are the parameters provided in `-brain` and `-label`. Additionally the output directory also contains a list of transformed thumbnails of similar

resolution as the input images. The thumbnails are saved as png images and are prefixed by the four digit sequence number so that they can be sorted by filenames and displayed in succession.

For example, for an input command

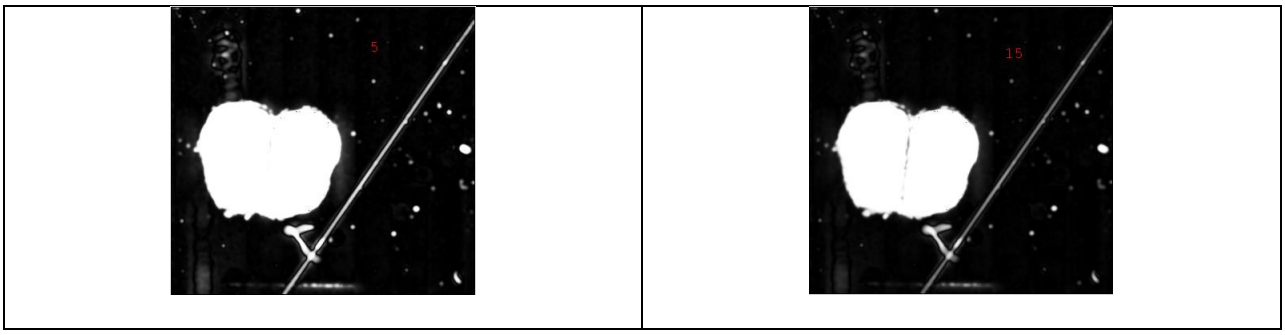
```
./StackAlign List_file.txt –brain JGA1 –label N –output myOutput
```

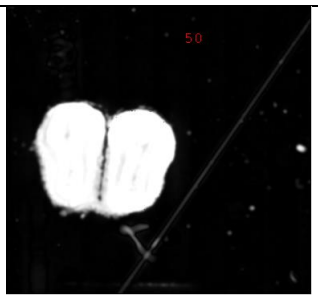
a typical output directory would contain the following files

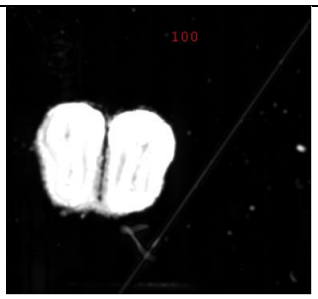
```
ls myOutput/
DIFF
JGA1_N.txt
JGA1_N_XForm.txt
0000_JGA1-N38-2013.04.06-03.28.16_JGA1_3_(0)0114.png
0001_JGA1-N39-2013.04.06-03.46.29_JGA1_1_(0)0115.png
0002_JGA1-N39-2013.04.06-03.46.29_JGA1_2_(0)0116.png
0003_JGA1-N39-2013.04.06-03.46.29_JGA1_3_(0)0117.png
0005_JGA1-N40-2013.04.06-04.04.33_JGA1_2_(0)0119.png
0006_JGA1-N40-2013.04.06-04.04.33_JGA1_3_(0)0120.png
0007_JGA1-N41-2013.04.06-04.22.43_JGA1_1_(0)0121.png
…..
```

Note the first 4 digits and the "_" is prefixed automatically to the input file names.

`-beta_inv` parameter controls the color to gray-scale conversion of the input images. Note that this is not a critical parameter and can be left unspecified. Figure 1 shows grayscale converted image for `beta_inv = 5 (top left), 15 (top right), 50 (bottom left) and 100 (bottom right).`







-gx and -gy is the canvas size, which loosely influences the maximum amount of translation that is possible. The default value of 1000 should be adequate for most of the images at the resolution we use. However, if any of the input image size is greater than 1000, then this parameter.

b. For *StackRepair*, the arguments are

```
[amit@mitragpu2 build]$ ./StackRepair

Usage : ./StackRepair OLDOutputDirectory NEWOutputDirectory [ options ]
Options:
        -ROT (global rotation in degrees, default: 0)
        -X (global translation X in pixels, default: 0)
        -Y (global translation Y in pixels, default: 0)
        -gx (canvas size X in pixels, default: 1000)
        -gy (canvas size Y in pixels, default: 1000)
        -cmd (repair string in [], e.g. [5:6,7,5c0.1,6r-25])
```

In this case OLDOutputDirectory and NEWOutputDirectory are both mandatory and they cannot be the same directory. Note that *StackRepair* only reads from OLDOutputDirectory and writes contents to NEWOutputDirectory after deleting its contents completely. All other arguments change the stack and are optional (i.e. if you call *StackRepair* with only two arguments and no other parameters, then the contents of OLDOutputDirectory are replicated in NEWOutputDirectory.)

-ROT applies a global rotation to the entire stack, -X and -Y applies translation (in pixels). -gx and -gy are same as before and can be different from the values used in StackAlign.

`-cmd` is a repair command string without any white-space. It must start with a "[" and end with "]" and each entry must be separated by a ",". There can be two kinds of entries in the repair string, X and X:Y, where X and Y are sequence number of the image files. When the entry is a single sequence number X, then *StackRepair deletes* that image from the stack and recomposes the stack. Then the entry contain two sequence numbers separated by a ":" (colon)[4], like X:Y then the transformation is re-estimated with a more robust criteria. Example of `-cmd` is `[8,9:10,56,221:222]` where 8 and 56 sequence numbers are deleted and the transformation between sequence number 9 and 10, and 221 and 222 are re-estimated. Note that the as a result of deleting 8 and 56, a new link is needed between 7:9 and 55:57 which are automatically determined and estimated.

Two additional commands were added to the two existing commands to cover some of the difficult cases. These commands are "pre-rotation" and "border cropping". Pre-rotation can be specified for any section using the symbol "r", e.g. [5r-45,5:6] means that section 5 will be pre-rotated by -45 degrees and then the link between 5 and 6 will be re-estimated. Note that the other link for section 5 with the previous section remains unchanged. Prerotation essentially initializes the optimizer with a specific rotation angle, so that a local minima in the vicinity of the prerotation angle can be found. Border-cropping is a similar tool that removes the outer border by a fraction of the total length of the side. It can be specified as [5c0.2,4:5], which means that 0.2 x side is cropped from image, i.e. 0.1xheight is cropped from top and bottom margins respectively and 0.1xwidth is cropped from left and right margins respectively for section 5. Then, the link between 4 and 5 is estimated. As before, the link between 5 and the next section remains unchanged.

Two important points about pre-rotation and border cropping: First, you must specify a link with these commands, otherwise there is no use doing these operations if the link is not

[4] Note that the "-" in the earlier version is replaced by ":" to avoid confusion of "-" which also implies negative sign used in other arguments.

estimated. Second, you may specify pre-rotation or border cropping or *both* for one or both the sections for a link. Since these are independent operations, they can be mixed without interfering. In other words, [12c0.1,12r50,13c0.2,13r-45,12:13] is a valid and legal command.

ExecuteBatchMultiple works as a job manager. It takes in as input the `brainName_labelName_XForm.txt` produced by *StackAlign* or *StackRepair*.

```
Usage: ./ExecuteBatchMultiple SectionListFile [batchsize]
```

The `batchsize` parameter determines number of parallel jobs that are submitted. This argument can be left as default, which submits groups of 10 sections as 30 batches is adequate for series having upto 300 sections. If the number of sections are greater than 300, then batchsize should be set to a value greater then N/10, where N is the total number of sections and 10 is the group_size empirically determined for optimal performance.

Additional utilities:

I have added a few utility codes specific to MBA project that interacts with the MBAStorageDB. Note that this code is intentionally decoupled form MBAStorageDB database because of inconsistencies in it. The following two utilities would help to interact with the MBAStorageDB.

1) FindDuplicateSections – This utility helps in determining duplicate sections in the MBAStorageDB. A duplicate section is defined as two sections of the same brain and label with identical ModeIndex. If a duplicate section exist, it becomes ambiguous to the registration code, and it might create confusion during transformation or other processing which section was used. So it is recommended to eliminate Duplicate Sections completely before processing the brain for registration.

   The code without any argument would search and flag Duplicate sections in the entire database. It modifies the isDuplicate flag in the Navigator_sections table.

The code with one or more brain name(s) as argument processes and updates only those brains. Example usage

```
./FindDuplicateSections PMD1234
```

2) CreateBrainList – This utility creates an initial brainlist using MBAStorageDB. This convenient utility works for project brains and those brains that have database entry in a project style filename. It runs a standard query and creates a list of sections for each label type in a text file. This text file can be used as input to the StackAlign code directly. The standard database query implemented here is

```
SELECT CONCAT(path,filename) FROM Navigator_section WHERE brain_id=BrainName AND label=labelName AND mislabeled=0 AND isDamaged=0 AND reCoverslip=0 AND reimage=0 AND goDelete=0 ORDER BY modeIndex;
```

Note that sometimes the path in the database does not correspond to the path of the lossless jp2 image file. This problem will likely be fixed. The logic used to detect the correct location of jp2 files is also implemented here. Therefore, it is more accurate to generate the list of section paths using this utility than a direct database query above. For any other query combination, or method for generating the brain list, this utility should not be used.