Amit Mukherjee

Mitralab, CSHL

5 September 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.

StackRepair – This module fixes/repairs some of the errors 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.

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 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. It is expected that the user will query database and create a list of input images for stack generation. 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 0th 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 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.
   d. If any of the lines containing image file path is not valid, the software will halt expecting user intervention. The user can either stop the process and correct the

input file OR he can investigate why the image is missing (e.g. M drives not accessible).

3. *StackRepair* works on the output of *StackAlign.* Note that the software package attempts to create stacks robustly from any set of images. Occasionally, these images may contain tissue / imaging damages or bad cropping. Since this is a sequential chain, the weakest link in the chain determines the quality of alignment. *StackRepair* provides a tools to repair or correct a broken link in the chain. 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.
    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 that is applied. One purpose of doing a *StackRepair* instead of redoing *StackAlign* (with the bad image removed) is saving time, because a 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 making the process faster.

4. 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 image file names to be stacked. It is a mandatory argument. Other mandatory arguments are `–brain` and `–label`. Note that since this software does not interact with the database, 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 written. Note that `StackAlign` outputs two text files (containing parameters) and transformed thumbnails for all the valid images in the `List_file`. 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 name and displayed in succession.

For example, for an input command

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

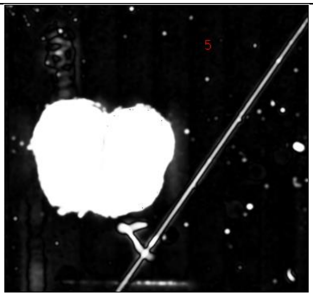
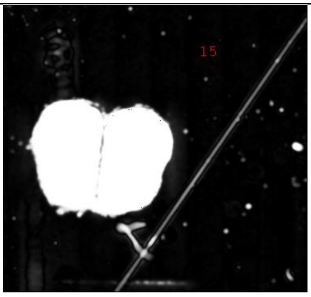
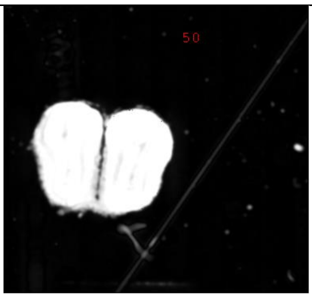
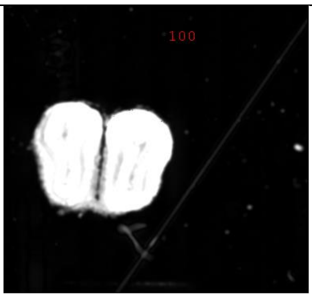
a typical output directory would contain the following files

```
ls myOutput/
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 touched. 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 ever 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 should be specified with a value greater than the largest image size.

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])
```

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 old contents completely. User should not get confused by the name used, only one directory is the real output (sorry ☺). All other arguments are optional. If one calls *StackRepair* with only two arguments and no other parameters, then the contents of `OLDOutputDirectory` are replicated in `NEWOutputDirectory`. The stack changes when these arguments are specified.

`-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` takes as input a repair command string without any white-space. The repair command string 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. When the entry contain two sequence numbers separated by a "-" (dash), 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.

c. *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.