GLISTR Software Manual

Version 3.0.0

Dongjin Kwon Ali Gooya Andreas Schuh

Contents

GLioma Image SegmenTation and Registration (GLISTR) [TMI2012], [MICCAI2014] is a software package designed for simultaneously segmenting brain scans of glioma patients and registering these scans to a normal atlas.

Some typical applications of GLISTR include,

- Labeling entire brain regions of glioma patients;
- Mapping gliomas into the healthy atlas space;
- Estimating parameters of the tumor growth model.

GLISTR is implemented as a command-line tool. It is semi-automatic and requires minimal user initializations. Users could use the visual interface called BrainTumorViewer to easily make initializations and a script for the execution. As a results, GLISTR will output a label map, a mapping between atlas and input, tumor parameters, etc.

1 About the algorithm

Computerized analysis of glioma images can be clinically important for different reasons. Segmentation of multi-channel MR glioma images into different tissue classes might be insightful for neuro-surgical interventions as surgeons aim to resect the most of tumor while minimizing the risk of damaging the healthy surrounding tissues. However, manual tracing of tumor borders can be a very tedious task and might be inaccurate due to the errors introduced in border delineation process. Therefore, a reliable automatic segmentation algorithm can be helpful to provide labelings of the patient tissues and facilitate the subsequent expert editing.

On the other hand, deformable registration of brain tumor images allows us to map data from different brain tumor patients into a common stereotaxic space, thereby enabling us the construction of statistical brain tumor atlases. These atlases are based on collective morphological, functional, and pathological information and increase our understanding of the Glioma's specific behaviour in various regions of the brain. For example, Glioma atlases are used for tasks such as learning the relative location of tumors with respect to the tumor grades.

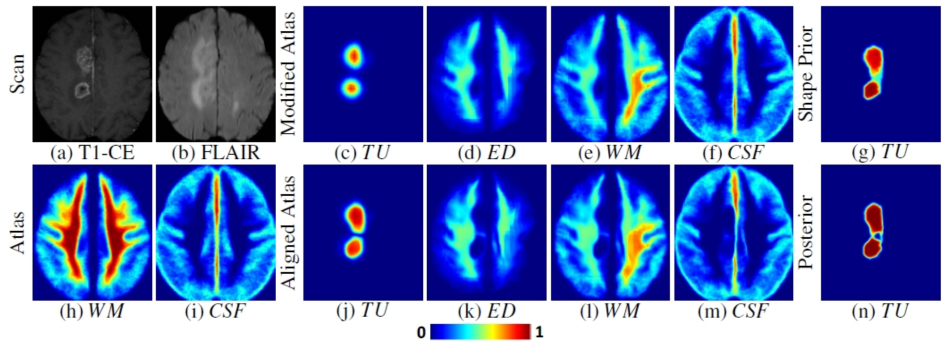


Figure 1: An example of spatial probabilities for multifocal glioma. We show subject scans in (a)-(b), normal atlas in (h)-(i), spatial probabilities obtained by growing tumors on normal atlas in (c)-(f), tumor shape prior in (g), spatial probabilities aligned to the scan in (j)-(m), and tumor posterior in (n).

In Figure 1, an example of spatial probabilities for multifocal glioma obtained by GLISTR is shown. The spatial probabilities in (c)-(f) are modified from the atlas shown in (h)-(i) using the tumor growth model. The tumor shape prior is shown in (g) computer obtained applying the random walk with restart on the subject scan shown in (a)-(b). This shape prior shows high probability values in the tumor region. Aligned spatial probabilities are shown in (j)-(m) optimized via the EM algorithm. These spatial probabilities and the tumor posterior shown in (n) fit well to healthy tissue and pathological regions shown in (a)-(b).

GLISTR requires minimal user initializations including seed points and radius for each tumor and one sample point for each tissue class. Users could use the visual interface called BrainTumorViewer to easily mark each point. Another input to GLISTR consists of the preprocessed patient scans and a list of probabilistic atlas priors representing a healthy population (we use jakob or eve currently). For preprocessing, we coregistered all modalities, corrected MR field inhomogeneity, skull stripping, and scaled intensities to fit [0, 255]. The output of GLISTR consists of a label map corresponding to the patient scans, deformation field(s) warping the originally normal atlas into the patient space, and the estimated tumor model parameters.

The current implementation of GLISTR accepts four MR modalities: T1, T1-CE, T2, and FLAIR. It segments these images into 10 labels: cerebrospinal fluid (CSF), gray matter (GM), white matter (WM), vessel (VS), edema (ED), necrosis (NCR), enhancing tumor (TU), non-enhancing tumor (NE) (optional), cerebellum (CB), and background (BG). We also define NT as the union of NE and NCR.

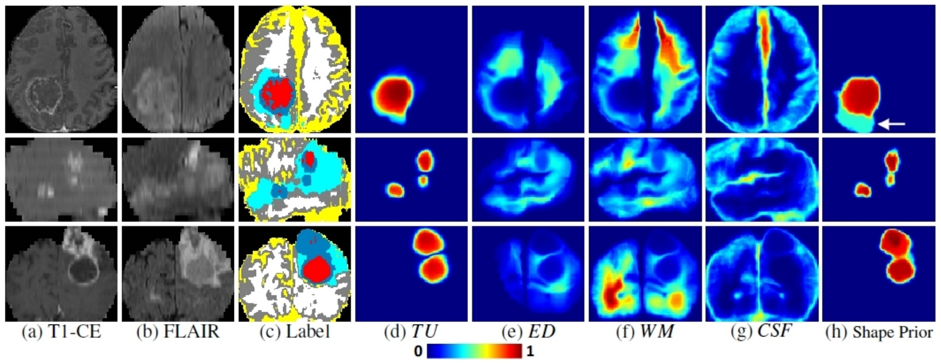


Figure 2: Segmentation and registration results for 3 subjects selected from the BRATS 2013 leaderboard data set. The top 1 row shows a single-focal glioma and bottom 2 rows show multi-focal gliomas. We show subject images in (a)-(b), segmentation results in (c) (indicating ET, NT, ED, WM, GM, and CSF in blue, red, cyan, white, gray, and yellow colors, respectively), spatial probabilities in (d)-(g), and tumor shape priors in (h).

In Figure 2, segmentation and registration results of GLISTR selected from the BRATS 2013 leaderboard data set are displayed. The top 1 row shows a single-focal glioma and bottom 2 rows show multi-focal gliomas. The subject images are shown in (a)-(b) and segmentation results are shown in (c) (indicating ET, NT, ED, WM, GM, and CSF in blue, red, cyan, white, gray, and yellow colors, respectively). The spatial probabilities aligned to the scan in (d)-(g) show that they fit well to the scan. The tumor shape priors in (h) help to align spatial probabilities for tumor as they initially estimate tumor regions reasonably well. However, GLISTR also showed robust estimation of tumor regions when the shape prior leaked into nearby regions as indicated by arrow in the fourth row of (h). The spatial probabilities for tumor were not expanded to this leaked region as the image likelihood for tumor kept lower values than those of healthy tissues on this region during the EM iterations. As a result, segmentations in (c) show visually reasonable tissue estimation especially for ET, NT, and ED regions.

2 Download

2.1 Software License

The GLISTR software is freely available under a BSD-style open source license that is compatible with the Open Source Definition by The Open Source Initiative and contains no restrictions on use of the software. The full license text is included with the distribution package and available online.

2.2 Documentation

GLISTR Software Manual: The software manual of GLISTR in PDF.

2.3 System Requirements

Operating System: Linux, Windows (64 bit)

Memory Requirement: 12GB or more.

2.4 Register for Download

Please **register online** to receive an email with the download links of the software.

3 Installation

3.1 Build Dependencies

Before building GLISTR, the following software libraries are required te installed.

Package	Version	Description
CMake	2.8.4 or higher	To compile and build GLISTR. Use version 2.8.4 or higher.
ITK	3.14 or higher	For ITK 3.x, `ITK_USE_REVIEW` option is required to set `ON`. For ITK 4.x, there's no mandatory option. In Windows, build ITK using CMake with the Win64 (x64) solution platform of Visual Studio.
LAPACK	3.2.1	Used for solving linear systems in GLISTR. In Windows, build CLAPACK using CMake with the Win64 (x64) solution platform of Visual Studio.

Note: To build in Windows, use CMake with the Win64 (x64) solution platform of Visual Studio. Our testing environment is Windows 7 64-bit and Visual Studio 2010.

3.2 Runtime Requirements

For the successful execution of GLISTR, the following software packages have to be installed.

Dependency	Version	Description
FSL (FLIRT)	3.3.11 – 5.0.7	GLISTR uses FLIRT for the affine alignment. For Linux and Mac OS, installation packages are provided. In Windows, it could be built from the patched source codes on the Cygwin environment. See the how-to guides *How to build FLIRT in Windows*.
HOPSPACK	2.0.2	Used for the optimization of the tumor growth model parameters. The mutithreaded version is used. Download and install the precompiled binary.
BrainTumorModeling_CoupledSolver	1.2.0 or higher	Used for the tumor growth simulation. Note that this package depends on the PETSc library. In Windows, it could be built on the Cygwin environment. For details on the build of this tumor simulator, see the how-to guides *How to build Tumor Simulator*.
BrainTumorViewer	1.0.0	Used for making initializations of GLISTR. It is an **optional** program and not required for executing GLISTR. For details on the build of this viewer, see the how-to guides *How to build BrainTumorViewer*.

To build BrainTumorModeling_CoupledSolver, see *How to build Tumor Simulator*. To build FLIRT in Windows, see *How to build FLIRT in Windows*. To build BrainTumorViewer, see *How to build BrainTumorViewer*. For HOPSPACK, copy the `HOPSPACK_main_threaded` executable to the appropriate directory which could be found by the `PATH` environment variable or GLISTR's installation `bin` folder.

3.3 Build and Installation

Please follow commands below in a shell/terminal (e.g., Bash). They will configure and build GLISTR using GNU Make (or Visual Studio in Windows). The main CMake configuration file (`CMakeLists.txt`) is located in the root directory of the package.

Step 1. Extract source files and create the build directory:

```
tar xzf glistr-${version}-source.tar.gz
mkdir glistr-${version}-build
cd glistr-${version}-build
```

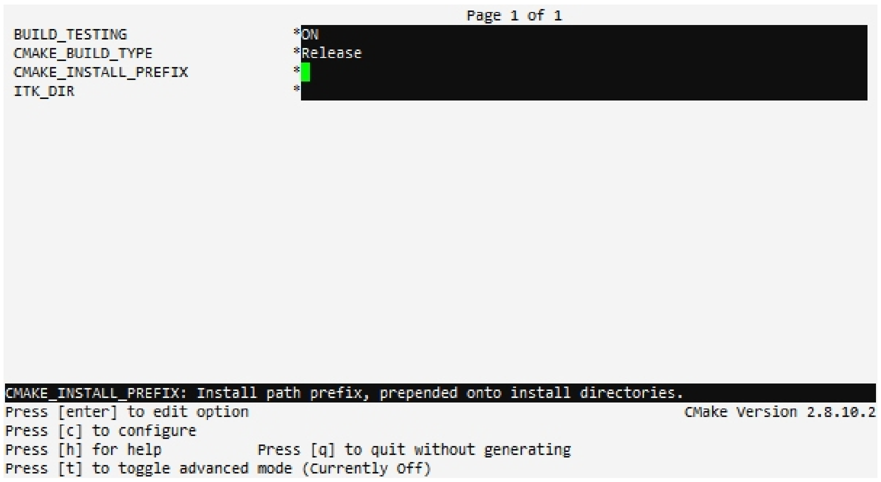
Note: In Windows, use the appropriate zip program (e.g., 7-zip) to extract.

Step 2. Run CMake to configure the build tree:

```
ccmake ../glistr-${version}-source
```

After the execution of this command and proceed the configure once, you will see a screen like Figure 3.

Note: In Windows, use the CMake GUI instead and set the generator as Visual Studio (Win64).



```
                                   Page 1 of 1
 BUILD_TESTING                    *ON
 CMAKE_BUILD_TYPE                 *Release
 CMAKE_INSTALL_PREFIX             *
 ITK_DIR                          *


CMAKE_INSTALL_PREFIX: Install path prefix, prepended onto install directories.
Press [enter] to edit option                                   CMake Version 2.8.10.2
Press [c] to configure
Press [h] for help           Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```

Figure 3: Configuring GLISTR installation using `ccmake`.

In the CMake interface, follow these steps:

2.1. Change `CMAKE_INSTALL_PREFIX` to the folder you want to install GLISTR into. This folder should be outside the `glistr-${version}-source` folder. Make sure you have the **write** access to this folder. Change `CMAKE_BUILD_TYPE` (or `CMAKE_CONFIGURATION_TYPES`) to `Release`. Change `ITK_DIR` to the folder that ITK is installed if it fails to find it.

Note: In Windows, also change `LAPACK_INCLUDE` and `LAPACK_LIB` to the folder that LAPACK is installed it fails to find them.

2.2. Keep pressing letter `c` on your keyboard until option `g` is available/displayed on the screen.

2.3. Then press `g` on your keyboard to generate the makefiles and to quit this ccmake window.

Note: In the CMake GUI, `c` and `g` correspond to `Configure` and `Generate` buttons.

Step 3. Build:

```
make
```

Note: In Windows, launch the solution file of Visual Studio, select `Release` in the solution configurations (and confirm `x64` in the soultion platforms), and then perform the rebuild solution.

Step 4. Test (optional):

```
make test
```

Note: In Windows, build the `RUN_TESTS` project. To perform tests, the `BUILD_TESTING` option in the CMake configuration is required to set `ON`.

In case of failing tests, re-run the tests, but this time by executing CTest directly with the `-V` option to enable verbose output and redirect the output to a text file:

```
ctest -V >& glistr-test.log
```

And send the file `glistr-test.log` as attachment of the issue report to `sbia-software at uphs.upenn.edu`.

Step 5. Install:

```
make install
```

Note: In Windows, build the `INSTALL` project.

Upon the success of the above compilation and build process, GLISTR is installed into the directory specified by the `CMAKE_INSTALL_PREFIX` (set during build configuration in Step 2). The GLISTR *Command-line Tools* are located in the `bin` subdirectory. Also, confirm the following programs could be found in the path or GLISTR's `bin` folder: `flirt`, `convert_xfm`, `HOPSPACK_main_threaded`, and `ForwardSolverDiffusion`. If not, check *Runtime Requirements* again.

How to build FLIRT in Windows

FLIRT is a part of FSL and it is not distributed in Windows. Here, we describe how to build FLIRT from the source code and run FLIRT in the command line in Windows. As we compile FLIRT on the Cygwin environment, install Cygwin before starting (confirm gcc version 4 is installed). After that, download FSL 3.3.11 source codes from here, and copy source code patch file located in `/lib/fsl-3.3.11-sources-flirt-cygwin.patch` of the package.

then follow the steps below in the Cygwin terminal.

Step 1. Extract source files:

```
tar xzf fsl-3.3.11-sources.tar.gz
```

Step 2. Apply patch:

```
patch -d fsl -p1 < fsl-3.3.11-sources-flirt-cygwin.patch
```

Note: This patch only works on the source codes relevant to FLIRT. Other FSL modules may need additional patches.

Step 3. Change to the fsl folder and set environmental variables:

```
cd fsl
export FSLDIR=$PWD
export FSLDEVDIR=${FSLDIR}
export FSLCONFDIR=${FSLDIR}/config
export FSLMACHTYPE=`gcc -dumpmachine`-gcc`gcc -dumpversion`
```

Make sure gcc is linked to the gcc-4 binary.

Step 4. Copy config file:

```
mkdir ./config/`gcc -dumpmachine`-gcc`gcc -dumpversion`
cp ./config/i686-pc-cygwin-gcc3.4.4/* ./config/`gcc -dumpmachine`-gcc`gcc -dumpversion`
```

Step 5. Build:

```
./build newmat utils znzlib niftiio fslio miscmaths newimage flirt
```

Step 6. Install

Upon the success of the above build process, copy `fsl/bin/flirt.exe` and `fsl/bin/convert_xfm.exe` executables into the Cygwin's `bin` folder (we assume it located in `C:\cygwin\bin`.). Also, add the Cygwin's `bin` folder in the system path if it is not included:

```
setx Path "%Path%;C:\cygwin\bin" /m
```

If users don't want to modify the path variable, copy `fsl/bin/flirt.exe` and `fsl/bin/convert_xfm.exe` executables into GLISTR's installation `bin` folder (where `GLISTR.exe` is installed). Also, copy `cygwin1.dll` and its dependent dlls (in our case, `cyggcc_s-1.dll`, `cyggfortran-3.dll`, `cygICE-6.dll`, `cygSM-6.dll`, `cygstdc++-6.dll`, `cyguuid-1.dll`, `cygX11-6.dll`, `cygXau-6.dll`, `cygxcb-1.dll`, `cygXdmcp-6.dll`, `cygXt-6.dll`) into that folder. Before running `flirt` in the command line, the following environmental variable is needed to be set.:

```
set FSLOUTPUTTYPE=NIFTI_GZ
```

How to build Tumor Simulator

To build BrainTumorModeling_CoupledSolver (BTMCS), please follow the installation_procedure_of_BTMCS. The source code could be obtained via the download URL. The download URL for the BrainTumorModeling_CoupledSolver software is sent to you per email in response to your software request for GLISTR. An email with the download links of the package can be requested by filling this online form with the appropriate information.

Upon the success of the build process, copy the `ForwardSolverDiffusion` executable to the appropriate directory which could be found by the `PATH` environment variable or GLISTR's installation `bin` folder.

How to build BrainTumorViewer

To build BrainTumorViewer, please follow the installation_procedure_of_BrainTumorViewer. The source code could be obtained via the download URL. The download URL for the BrainTumorViewer software is sent to you per email in response to your software request for BrainTumorViewer. An email with the download links of the package can be requested by filling this online form with the appropriate information.

4 Manual

4.1 GLISTR Command and Parameters

The main command of GLISTR which segments the subject scan and registers the atlas to the subject scan is named `GLISTR`. The simplest use is:

```
GLISTR  --featureslist <list_file>  --seed_info <seed_file>  --point_info <point_file>
        --atlas_folder <atlas_folder>  --outputdir <output_folder>
```

The parameters used above are **mandatory**. For image format, NIfTI-1 (`*.nii` or `*.nii.gz`) format is required. For memory requirements, it depends on the size of the atlas and the number of thread used. For the eve atlas with using single thread, 12GB memory is recommended. For 3 threads in same configuration, 15GB memory is recommended. The absolute path is required for each path in the parameter. The meaning of each parameter is:

```
--featureslist, -fl <file>    Text file listing the patient scans, one per line.
                              It is recommended to use T1, T1-CE, T2, and FLAIR patient
                              scans as input.

--seed_info, -si <file>       Tumor seed information. This option overrides -x, -y, -z,
                              and -T. The format of this file is:

                                  <n: number of seeds>
                                  <x1> <y1> <z1> <r1>
                                  ...
                                  <xn> <yn> <zn> <rn>

                              where <xi> <yi> <zi> is the RAS coordinate of ith seed and
                              <di> is the approximated diameter of ith tumor.

--point_info, -pi <file>      One sample point for each class. This option overrides -im.
                              The format of this file is:

                                  <Class 1>
                                  <x1> <y1> <z1>
                                  ...
                                  <Class n>
                                  <xn> <yn> <zn>

                              where <Class i> is the name of ith tissue class and
                              <xi> <yi> <zi> is the RAS coordinate of ith tissue class.

--atlas_folder, -af <dir>     The directory path for the atlas.
--outputdir, -d <dir>         The directory path for output files.
```

Each parameter can be specified as a long name starting with `--` or a short name starting with `-`. For tumor seed information, users can specify the number of seed points up to 10. For example, if there're two tumor masses in the scan, the tumor seed file contains:

```
2
x1 y1 z1 d1
x2 y2 z2 d2
```

where <xi> <yi> <zi> is the RAS coordinate of ith center of the tumor mass and <di> is its approximated diameter.

For each tissue class, the label index and value are defined as follows:

Label	Description	Index	Value
BG	Background	0	0
CSF	Cerebrospinal Fluid	1	10
GM	Gray Matter	2	150
WM	White Matter	3	250
VS	Vessel	4	25
ED	Edema	5	100
NCR	Necrosis	6	175
TU	Enhancing Tumor	7	200
NE	Non-Enhancing Tumor	8	185
CB	Cerebellum	9	5

The NE label is optional and it is not segmented if not specified in the point file. The point file contains the location of one sample point for each tissue class. The typical point file contains:

```
CSF
x1 y1 z1
GM
x2 y2 z2
WM
x3 y3 z3
VS
x4 y4 z4
ED
x5 y5 z5
NCR
x6 y6 z6
TU
x7 y7 z7
CB
x9 y9 z9
```

where <xi> <yi> <zi> is the RAS coordinate of one sample point for ith tissue class. The point for BG doesn't need to be specified.

The following parameters are **optional**:

```
--workingdir, -w <dir>            The directory path for intermediate files.
--atlas_template, -at <file>      The atlas template file.
--atlas_prior, -ap <file>         The list file for atlas priors.
--atlas_label, -al <file>         The atlas label map (.nii.gz).
--atlas_label_img, -ali <file>    The atlas label map (.img).
--num_omp_threads, -not <int>     The number of OpemMP threads.
--num_itk_threads, -nit <int>     The number of ITK threads.
--num_hop_threads, -nht <int>     The number of HOPSPACK threads.
--verbose, -v <int>               if greater than 0, print log messages:
                                  1 (default), 2 (detailed).
```

The atlas related parameters (`--atlas_template`, `--atlas_prior`, `--atlas_label`, and `--atlas_label_img`) are required to be specified all together and they are effective only when `--atlas_folder` is not specified. The following parameters are used in the previous version, however they became **obsolete** in this version:

```
--xc, -x <float>             Initial tumor center in sagital direction.
--yc, -y <float>             Initial tumor center in coronal direction.
--zc, -z <float>             Initial tumor center in axial direction.
--growthtime, -T <int>       Initial tumor growth length. Should be larger than 25.
                             For huge tumors, set it to about 120.
--initialmeans, -im <file>   Text file specifying the initial mean values for
                             the different classes.
```

4.2 Output of GLISTR

The following output files are generated by GLISTR in the `<output_folder>` specified by the `--outputdir` or `-d` parameter:

scan_label_map.nii.gz	Obtained segmentation label map.
scan_to_atlas.mat	Affine matrix mapping the patient scan to the atals.
atlas_to_scan.mat	Inverse matrix of scan_to_atlas.mat.
scan_u_h_field.*	Deformation field mapping affine transformed patient scan to atlas.
tumor_params.txt	Obtained tumor parameters.
scan_means.txt	Obtained mean value for each tissue class.
scan_variances.txt	Obtained variance value for each tissue class.
scan_posterior*	Obtained posterior map for each tissue class.
scan_prior*	Obtained prior map for each tissue class.
scan_init_menas.txt	Initial mean value for each tissue class.
log.txt	Log file.

To map an image in the atals space to the patient space, warp the image using `scan_u_h_field.mhd` first and then apply affine transform using `atlas_to_scan.mat`. `scan_posterior*` and `scan_prior*` for each tissue class are appended with the label index defined in the above table. Also, the label value of each class used in `scan_label_map.nii.gz` is defined above.

4.3 Making Initializations Using BrainTumorViewer

Using BrainTumorViewer, making initial tumor seeds and tissue points could be easily done. Assuming co-registered scans for brain tumor MRI are given, open all modalities using `File > Open Image(s)` menu or drag and drop images to the viewer. Users can navigate each modality and roughly figure out the location of the tumor and the edema. In the `Tumor` tab as shown in Figure 4, select `GLISTR` in `Type of Initializations`. To add a point for tumor seed, click one item in the `Tumor Seed Points` list and then click (move cross hair to) an approximate center of each tumor mass and push `< space >`. To add a radius for this mass, click its approximate boundary and push `< shift > + < space >`. After completing, the list can be saved using the Save button. In the `Tissue Points` list, each row represent one sample location for the corresponding tissue type. To add a point, select one tissue type and then click one sample location of that tissue on the view and push `< space >`.

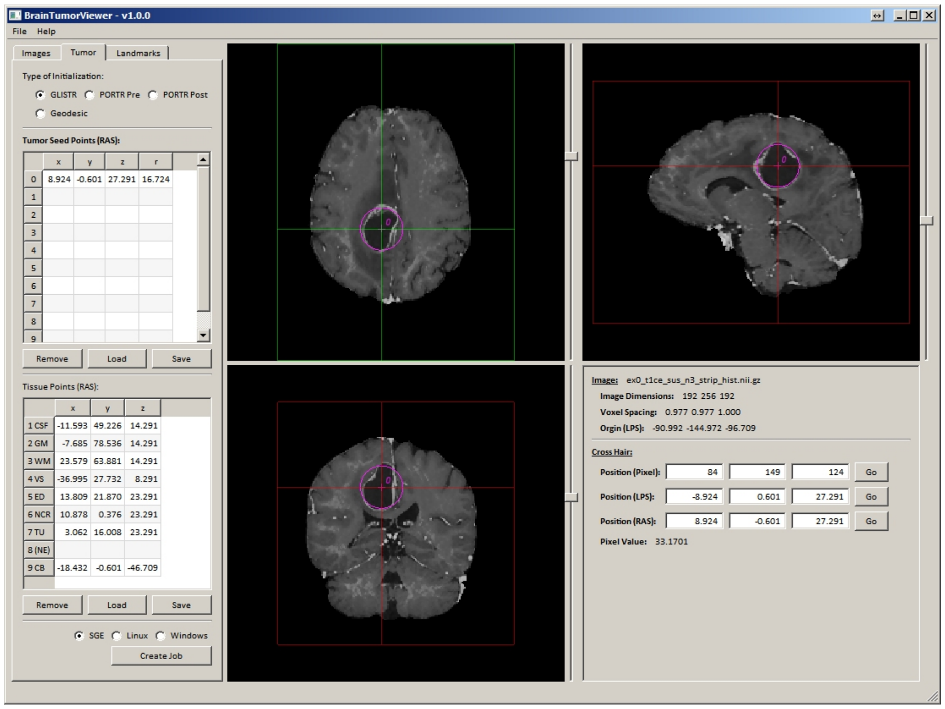
4.4 Auxiliary Commands

Command	Operation
WarpImage{NN}	Warp image
NormalizeImage	Normalize image
ResampleImage	Resample image
ConcatenateFields	Concatenate two deformation fields
ResampleDeformationField	Resample deformation field
ReverseDeformationField	Reverse deformation field
EvaluateQ	Computing the cost function

`EvaluateQ` is the internal procedure and called only by `GLISTR`.

4.5 Running Example

If we assume `$(install)` is an installation directory of GLISTR, an example scans and its worked initializations are installed in `$(install)/example/ex0`. The GLISTR for this example is performed by



Type of Initialization: [✓ GLISTR] PORTR Pre PORTR Post Geodesic

Tumor Seed Points (RAS):

	x	y	z	r
0	8.924	-0.601	27.291	16.724
1				
2				
3				
4				
5				
6				
7				
8				
9				

Tissue Points (RAS):

	x	y	z
1 CSF	-11.593	49.226	14.291
2 GM	-7.685	78.536	14.291
3 WM	23.579	63.881	14.291
4 VS	-36.995	27.732	8.291
5 ED	13.809	21.870	23.291
6 NCR	10.878	0.376	23.291
7 TU	3.062	16.008	23.291
8 (NE)			
9 CB	-18.432	-0.601	-46.709

[✓ SGE] Linux Windows

Image: ex0_t1ce_sus_n3_strip_hist.nii.gz
Image Dimensions: 192 256 192
Voxel Spacing: 0.977 0.977 1.000
Orgin (LPS): -90.992 -144.972 -96.709

Cross Hair:

Position (Pixel):	84	149	124
Position (LPS):	-8.924	0.601	27.291
Position (RAS):	8.924	-0.601	27.291

Pixel Value: 33.1701

Figure 4: An example screenshot of BrainTumorViewer.

```
$(install)/bin/GLISTR  --featureslist $(install)/example/ex0/scan0.lst
                       --seed_info $(install)/example/ex0/init_seed_glistr.txt
                       --point_info $(install)/example/ex0/init_point_glistr_10a.txt
                       --atlas_folder $(install)/atlas_jakob_with_cere_type
                       --outputdir $(install)/example/ex0/mglistr
```

Then, output files are created in `$(install)/example/ex0/mglistr`. To check the intermediate results of the procedure specify the working directory by appending `--workingdir $(install)/example/ex0/mglistr/tmp`. For `--atals_folder`, `$(install)/` is automatically prepended when users specify the last directory name only (e.g. `atlas_jakob_with_cere_type` in this example). To check more detailed output messages, append `--verbose 2`. To execute GLISTR using `N` cores in the system, append `--num_omp_threads N`, `--num_itk_threads N`, and `--num_hop_threads N`. The running time of this example is about 2 hours using 3 cores on an Intel Core i7 3.4 GHz machine with Windows operating system having 32GB memory.

Note: Before running GLISTR, confirm `flirt`, `convert_xfm` of FSL, `HOPSPACK_main_threaded` of HOPSPACK, and `ForwardSolverDiffusion` of BrainTumorModeling_CoupledSolver could be found in the path or located in `$(install)/bin` folder. If not, check *Runtime Requirements* again.

Warp Image

Given the refernce image and the deformation field (mapping refernce to input spaces), `WarpImage` or `WarpImageNN` warps an input image to the space of the refernce image.

```
WarpImage{NN}  -i [input_image_file]   -r [reference_image_file]
               -o [output_image_file]  -d [deformation_field_file]
```

The meaning of each parameter is

```
-i  (--input    ) [input_image_file]        input (moving) image (input)
-r  (--reference) [reference_image_file]    reference image (input)
-o  (--output   ) [output_image_file]       output image (output)
-d  (--def_field) [deformation_field_file]  reference to input (moving)
                                            def. field (input)
```

`WarpImage` is using the bilinear interpolation, for warping finite value images like label maps, use `WarpImageNN` which uses the nearest neighbor interpolation.

Normalize Image

`NormalizeImage` normalize an input image using the histogram mathing with the given template image. It additionally performs the susan smoothing and the n3 correction if options are selected.

```
NormalizeImage  -i [input_image_file]     -o [output_image_file]
                -t [template_image_file]  -a [template_wm_range_a]
                -b [template_wm_range_b]  -s [1 or 0]  -n [1 or 0]
```

The meaning of each parameter is

```
-i  (--input        ) [input_image_file]        input image (input)
-o  (--output       ) [output_image_file]       output image (output)
-t  (--template     ) [template_image_file]     template image (input)
-a  (--template_wm_a) [template_wm_range_a]     template wm range a, intensity in
                                                [a,b] is used for scaling
-b  (--template_wm_b) [template_wm_range_b]     template wm range b, intensity in
                                                [a,b] is used for scaling
-s  (--apply_susan  ) [1 or 0]                  1 (default) if apply susan smoothing,
                                                0 otherwise
-sb (--susan_bt     ) [bt]                      bt for susan
-sd (--susan_dt     ) [dt]                      dt for susan
-n  (--apply_n3     ) [1 or 0]                  1 (default) if apply n3 bias field
                                                correction, 0 otherwise
-sh (--scale_hist   ) [1 or 0]                  1 (default) if scale via histogram
                                                mathing, 0 scale using maximum value
-sm (--scale_max    ) [max]                     max value for scaling
-w  (--apply_window ) [1 or 0]                  1 (default) if apply windowing,
                                                0 otherwise
-wa (--window_a     ) [window_a]                window lower percent (default: 0.01)
-wb (--window_b     ) [window_b]                window upper percent (default: 0.09)
```

Resample Image

`ResampleImage` scales an input image using ratios in x, y, z axis. It requires the reference image in the resampled space.

```
ResampleImage  -i [input_image_file]  -o [output_image_file]
               -r [reference_image_file]
               -x [ratio_x] -y [ratio_y] -z [ratio_z]
```

The meaning of each parameter is

```
-i  (--input    ) [input_image_file]        input image file (input)
-o  (--output   ) [output_image_file]       output image file (output)
-r  (--reference) [reference_image_file]    reference image file (input)
-x  (--ratio_x  ) [ratio_x]                 ratio in x axis (input)
-y  (--ratio_y  ) [ratio_y]                 ratio in y axis (input)
-z  (--ratio_z  ) [ratio_z]                 ratio in z axis (input)
```

Concatenate Two Deformation Fields

`ConcatenateFields` concatenates two given deformation fields.

```
ConcatenateFields  -fi [fix_to_inter_deformation_file]
                   -im [inter_to_mov_deformation_file]
                   -fm [fix_to_mov_deformation_file]
```

The meaning of each parameter is

```
-fi (--fix_to_inter) [fix_to_inter_deformation_file]    fix to inter def. field (input)
-im (--inter_to_mov) [inter_to_mov_deformation_file]    inter to mov def. field (input)
-fm (--fix_to_mov  ) [fix_to_mov_deformation_file]      fix to mov def. field (output)
```

Resample Deformation Field

`ResampleDeformationField` scales an input deformation field using ratios in x, y, z axis. It requires the reference image in the resampled space.

```
ResampleDeformationField  -i [input_deformation_file]  -o [output_deformation_file]
                          -r [reference_image_file]
                          -x [ratio_x]  -y [ratio_y]  -z [ratio_z]
```

The meaning of each parameter is

```
-i  (--input    ) [input_deformation_file]      input def. field (input)
-o  (--output   ) [output_deformation_file]     output def. field (output)
-r  (--reference) [reference_image_file]        reference image (input)
-x  (--ratio_x  ) [ratio_x]                     ratio in x axis (input)
-y  (--ratio_y  ) [ratio_y]                     ratio in y axis (input)
-z  (--ratio_z  ) [ratio_z]                     ratio in z axis (input)
```

Reverse Deformation Field

`ReverseDeformationField` generates the inverse deforation field fron an input deformation field.

```
ReverseDeformationField  -i [input_deformation_file]  -o [output_deformation_file]
                         -n [iterations_number]  -s [stop_value]
```

The meaning of each parameter is

```
-i  (--input   ) [input_deformation_file]       input def. field (input)
-o  (--output  ) [output_deformation_file]      output def. field (output)
-n  (--num_iter) [number_iterations]            number of iterations (input)
-s  (--stop_val) [stop_value]                   stop value (input)
```

Higher iterations number give more accurate results. It stops the iteration if the error (in mm) between forward and backward mapping is smaller than specified `stop_val`.

5 Changelog

Version 3.0.0 (Nov 13, 2014)

- Fully reimplemented using C++ from the ground.
- Improved the performances by combining generative models.
- Enabled multithread option for tumor growths and optimizations.
- Allowed to place multiple tumore seeds for multi-focal gliomas.
- Changed to input locations for tissue points instead of input intensity values.
- Changed to use HOPSPACK from APPSPACK.
- Started to support **Windows**.
- Removed the dependency with BASIS.

Version 2.2.0 (Oct 30, 2012)

- Updated project to use BASIS 2.1.
- Add path to found `nifti` and `numpy` Python modules to `PYTHONPATH` in `python_launcher`.
- Fixed internal test of `glistr-sbia` command.
- Minor edit of GLISTR web pages.

Version 2.1.2 (Aug 3, 2012)

- Fixed typo in `src/CMakeLists.txt` which caused invalid/empty `exec_prefix` in `PYTHONHOM`.
- Added more details about imaging protocol for the acquisition of the input images used for evaluation.
- Added reference to most recent GLISTR TMI paper to software manual.

Version 2.1.1 (Jan 6, 2012)

- Enabled output of optimal tumor growth parameters, tumor density map, and tumor deformation field.
- Penalize tumors simulated in background.
- Bug fix in tumor simulator enables build of optimized code which reduces running time.
- Further decrease of running time by adaptive choice of maximum number of evaluations for optimization of tumor growth parameters.
- Ensure consistent results also when MPI version of APPSPACK is used.
- Added software tests which run in less than 10 minutes each.
- Enhanced build and installation instructions.
- Separated Slicer 4 module. An update of this module is prepared and will be released soon.

Version 2.1.0 (Feb 17, 2012)

- Enabled output of optimal tumor growth parameters, tumor density map, and tumor deformation field.

Version 2.0.0 (Dec 16, 2011)

- Completely refactored Python code and command-line interface to improve usability and maintenance.
- Includes Python module for direct import in Python code.
- Single convenience command named `glistr` which implements complete GLISTR processing pipeline.

- Optional build of GLISTR as Extension for Slicer 4. Extension implemented as Python scripted module with graphical user interface.

Version 1.0.0 (Nov 6, 2011)

- First public release of the GLISTR software.

6 Publications

Please cite [TMI2012] and [MICCAI2014] when you used this version of GLISTR in your research:

[MICCAI2014]
[TMI2012]
[MICCAI2011]
[TMI2011]
[JMB2008]

7 People

Advisor

- Christos Davatzikos ✉
- Kilian M. Pohl

Software Authors

- Dongjin Kwon ✉
  - Improved the method by combining generative models and applying multiple tumor seeds.
  - Developed the version 3 of software reimplemented using C++ from the ground.
- Ali Gooya
  - Initiated the project.
  - Developed the original version of software.

Contributors

- Andreas Schuh
  - Reviewed and revised the software package.
  - Managed the software upto the version 2.
- George Biros
  - Developed the tumor simulator used in GLISTR.

References

[MICCAI2014] D. Kwon, R.T. Shinohara, H. Akbari, C. Davatzikos, Combining Generative Models for Multifocal Glioma Segmentation and Registration, In: Proc. MICCAI (1): 763-770 (2014)

[TMI2012] A. Gooya, K.M. Pohl, M. Bilello, L. Cirillo, G. Biros, E.R. Melhem, C. Davatzikos, GLISTR: Glioma Image Segmentation and Registration, IEEE Trans. Med. Imaging 31(10): 1941-1954 (2012)

[MICCAI2011] A. Gooya, K.M. Pohl, M. Billelo, G. Biros and C. Davatzikos, Joint segmentation and deformable registration of brain scans guided by a tumor growth model, In: Proc. MICCAI (2): 532-540 (2011)

[TMI2011] A. Gooya, G. Biros, and C. Davatzikos, Deformable registration of Glioma images using EM algorithm and diffusion reaction modeling, IEEE Trans. Med. Imaging 30(2): 375-390 (2011)

[JMB2008] C. Hogea, C. Davatzikos, G. Biros, An image-driven parameter estimation problem for a reaction-diffusion glioma growth model with mass effects, J. Math. Biol. 56(6): 793-825 (2008)