Segmentation of the Ventricles Using the Insight Toolkit

Matthew Stickney

20 October 2007
stickm@rpi.edu

Abstract

This paper describes the use of the Insight Toolkit (www.itk.org) to segment the ventricles in MR image data obtained from the Designed Database of MR Brain Images of Healthy Volunteers hosted at the MIDAS project (http://insight-journal.org/dspace/).

Contents

Conditions for Reproducibility

While this experiment may be fully reproducible under other conditions, reproducibility is guaranteed under the following conditions:

- CMake 2.4
- Insight Toolkit 3.4
- MRI data from MIDAS

Also note that it is up to the user to download the data for this experiment. The specific data set used can be found here: http://hdl.handle.net/1926/659 [2] [1]. The file Normal012-T2.mha should be placed in the source directory with segmentation.cpp.

Reading the Code

The code in segmentation.cpp segments the ventricles in this data set through the use of itk's WatershedImageFilter class, following the use of a series of preprocessing filters. Afterward, and image is extracted using the ExtractImageFilter class.

The preprocessing for the image is done with 15 iterations of the GradientAnisotropicDiffusionImageFilter class, with a conductance of .5 and a timestep of .07. This is set in the code as follows:

```
smoothFilter->SetNumberOfIterations(15);
smoothFilter->SetConductanceParameter(.5);
smoothFilter->SetTimeStep(.07);
```

Next the smoothed image is converted to a height map using the GradientMagnitudeImageFilter class, with no special parameters set. This height function is passed into the WatershedFilter class, which is initialized with a threshold of .01 and a level of 0.2:

```
watershedFilter->SetThreshold(0.01);
watershedFilter->SetLevel(0.2);
```

Next, we connect the data pipeline from reader through the filters and out to a writer:

```
smoothFilter->SetInput( reader->GetOutput() );
gradientFilter->SetInput( smoothFilter->GetOutput() );
watershedFilter->SetInput( gradientFilter->GetOutput() );
writer->SetInput( watershedFilter->GetOutput() );
```

After calling `writer->Update()`, this outputs a file specified on the command line with the segmented image data. The next step is to produce a 2d image of a demonstrative slice from the segmented data; here we use the ExtractImageFilter class to capture a 2d slice of the segmented data at a demonstrative z value, setting the region and index appropriately:

```
region.SetIndex( sliceIndex );
region.SetSize( sliceSize );
extractFilter->SetExtractionRegion( region );
```

We then connect the data pipeline again, taking the output from watershedFilter:

```
extractFilter->SetInput( watershedFilter->GetOutput() );
sliceWriter->SetInput( extractFilter->GetOutput() );
```

`sliceWriter->Update()` is then called, writing the slice to a file (also specified on the command line).

Running the Code

To run this code, simply run Cmake, configure, generate the make files, then build and run the executable. While the program will technically run without a data set, it is strongly recommended that you download one to run the program on (if you wish to verify the author's exeperiment, then you should download Normal012-T2.mha from the above link). The usage for the program is Segmentation NameOfInputFile NameOfOutputFile NameOfSliceFile. Also note that while the parameters for the filters used are hardcoded into the program, they can be changed simply by editing the appropriate portions of segmentation.cpp and rebuilding (CMake does not need to to re-run for such small changes in a simple program).

The CmakeList.txt file does not define any tests on its own; however, it does build the ImageCompare executable found in the itk source tree. This utility can be used to compare the extracted slice with the baseline image provided by the author (note that the baseline was generated with default parameters and `Normal012-T2.mha;` you'll have to generate your own baseline image if you want this test to work on a different dataset). It is important to note that if the hardcoded filter values are changed, this comparison will fail even when run on the original data set (the original values are included in the comments in the code, in case of accidental change).

Results

The Watershed filter, in the end, was capable of segmenting the ventricles, but requires a great deal of care in selecting parameters to produce good segmentations. This filter also requires a good deal of preprocessing, and can be very intensive to run on detailed structures (the parameters involved can also affect this, such as having too low a threshold value). The Watershed filter also generates labeled regions in the final segmented data, so for visualization purposes it is also generally a good idea to use a colorizing filter on the watershed output (a filter to extract a certain labeled region from the data would also be useful for 3d visualizations). Overall, while the watershed filter has potential for high-quality segmentation, it is fairly complex to use properly, can require a great deal of time to test and adjust, and is therefore ill-suited to many segmentation applications, especially those where it may be desireable to introduce some level of automation into the process.

References

[1] Bullitt E, Zeng D, Gerig G, Aylward S, Joshi S, Smith JK, Lin W, Ewend MG (2005) Vessel tortuosity and brain tumor malignancy: A blinded study. Academic Radiology 12:1232-1240, 2005.

[2] Elizabeth Bullitt. Normal-012-t2. http://hdl.handle.net//1926/659, 2007.

[3] L. Ibanez, W. Schroeder, L. Ng, J. Cates. *The ITK Software Guide*. Kitware, INC. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, first edition, 2003.