Computing Textural Feature Maps for N-Dimensional images

Release 2.0.0

Jean-Baptiste Vimort[1], Matthew McCormick[1], François Budin[1] and Beatriz Paniagua[1]

July 6, 2017

[1]Kitware, Inc, Carrboro, NC

Abstract

This document describes a new remote module implemented for the Insight Toolkit ITK www.itk.org, itkTextureFeatures. This module contains two texture analysis filters that are used to compute feature maps of N-Dimensional images using two well-known texture analysis methods. The two filters contained in this module are itkCoocurrenceTextureFeaturesImageFilter (which computes textural features based on intensity-based co-occurrence matrices in the image) and itkRunLengthTextureFeaturesImageFilter (which computes textural features based on equally valued intensity clusters of different sizes or run lengths in the image). The output of this module is a vector image of the same size than the input that contains a multidimensional vector in each pixel/voxel. Filters can be configured based in the locality of the textural features (neighborhood size), offset directions for co-ocurrence and run length computation, the number of bins for the intensity histograms, the intensity range or the range of run lengths. This paper is accompanied with the source code, input data, parameters and output data that we have used for validating the algorithm described in this paper. This adheres to the fundamental principle that scientific publications must facilitate reproducibility of the reported results.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3574]

Contents

1 Introduction

Texture is an intuitive concept heuristically but difficult to define precisely. It can be defined as series of homogeneous visual patterns that are observed in certain kinds of materials. Humans describe texture through qualitative concepts such as fine, coarse, granulated or smooth. These descriptions are not precise and, in addition, they are not quantitative. The texture quantification has been studied for a long time and many different texture quantification and analysis techniques exist. The work presented here explores texture quantification algorithms that provide a statistical description of the local texture of a 3D image and produces N-Dimensional texture maps. More importantly, the work here presented allows to obtain these outputs close to real-time.

ITK already contains tools for texture analysis such as `itk::ScalarImageToTextureFeaturesFilter` or `itk::ScalarImageToRunLengthFeaturesFilter`. Each of these filters can be used in order to obtain a N-Dimensional texture characterization of an image.

However, those filters are implemented to describe images globally and they are not optimized to be used to describe local texture features. Using the existing filters to compute texture features iteratively for each pixels's neighborhood, in order to build texture maps, quickly becomes too time consuming. This is especially so when dealing with high resolution data, such as micro computed tomography (μCT) or on large local neighborhoods of the input image.

The chosen solution, described in this article, consists in creating a new ITK remote module (called itkTextureFeature) dedicated to the computation of feature maps for N-Dimensional images. The filters implemented in itkTextureFeature computes the exact same features as `itk::ScalarImageToTextureFeaturesFilter` and `itk::ScalarImageToRunLengthFeaturesFilter`. However, the new algorithms are optimized (particularly thanks to multithreading, `itk::NeighborhoodIterator`, `itk::ImageBoundaryFacesCalculator`) to be able to compute the feature maps much faster.

All the features available in itkTextureFeature are presented in Section 2. Section 3 describes the filters specifications (templates, inputs, parameters) of each filter and how to customize the use of these filters to each different texture analysis application. Section 4 contain examples of code using itkTextureFeature filters in Python and C++. Finally, Sections 5 and 6 present several scenarios, results and conclusions obtained with itkTextureFeatures.

2 Features Available

2.1 Co-occurence features

The computation of co-occurrence features is based on the grey level co-occurrence matrix (GLCM) computed with `itk::itkCoocurrenceTextureFeaturesImageFilter` for each pixel's neighborhood. The GLCM matrix describes intensity organization of each pixels' neighborhood thanks to its second-order joint probability function [6, 5, 2, 3]. In the following computations the nomenclature is as follows:

$g(i,j)$ is the element in the cell (i,j) of the normalized GLCM

$\mu = \sum_{i,j} i \cdot g(i,j) = \sum_{i,j} j \cdot g(i,j)$ is the weighted pixel average

$\sigma = \sum_{i,j} (i-\mu)^2 \cdot g(i,j) = \sum_{i,j} (j-\mu)^2 \cdot g(i,j)$ is the weighted pixel variance

μ_t and σ_t are the mean and standard deviation of the row (or column, due to symmetry) sums

`itk::itkCoocurrenceTextureFeaturesImageFilter` computes the following features from the co-occurrence matrix:

Energy is a feature that measures the local uniformity of texture. The higher the energy value, the bigger the uniformity and organization of the texture.

$$f_1 = \sum_{i,j} g(i,j)^2 \tag{1}$$

Entropy is a feature that expresses the level of organization of a texture. A completely random distribution of grey-level intensities in the image volume would have very high entropy, while an image with the same grey-level across all pixels would have very low value of entropy.

$$f_2 = \begin{cases} \sum_{i,j} g(i,j) log_2 g(i,j) & \text{if } g(i,j) \neq 0 \\ 0 & \text{if } g(i,j) = 0 \end{cases} \tag{2}$$

Correlation is a feature that measures the linear dependency of gray level values in the co-occurrence matrix.

$$f_3 = \sum_{i,j} \frac{(i-\mu)(j-\mu) g(i,j)}{\sigma^2} \tag{3}$$

Inverse Difference Moment (IDM) is a feature that measures the homogeneity of the image. IDM will be low for in-homogeneous images, and a higher value for homogeneous images.

$$f_4 = \sum_{i,j} \frac{1}{1+(i-j)^2} g(i,j) \tag{4}$$

Inertia or **contrast** is a feature that measures local grey-level variation in the GLCM matrix. If the neighboring pixels in the texture are very similar in their grey-level values then the contrast in the image is very low. Contrast is 0 for a constant image.

$$f_5 = \sum_{i,j} (i-j)^2 g(i,j) \tag{5}$$

Cluster Shade is a feature of the skewness of the matrix and is believed to be linked to perception of uniformity in the image. When this feature is high the image is asymmetric.

$$f_6 = \sum_{i,j} ((i-\mu)+(j-\mu))^3 g(i,j) \tag{6}$$

Cluster Prominence is a feature that is also related to the perceptual symmetry of the image. When the cluster prominence value is high, the image is less symmetric.

$$f_7 = \sum_{i,j}((i-\mu)+(j-\mu))^4 g(i,j) \tag{7}$$

Haralick's Correlation is the original correlation measure designed by Haralick in 1973, and measures the linear dependence between pixels relative to each other.

$$f_8 = \frac{\sum_{i,j}(i,j)g(i,j) - \mu_t^2}{\sigma_t^2} \tag{8}$$

2.2 Run length features

The computation of the run length features is based on the grey level run length matrix (GLRLM) computed inside `itk::itkRunLengthTextureFeaturesImageFilter` for each pixel's neighborhood. A grey-level run is a set of consecutive, collinear picture points having the same grey-level value. The length of the run is the number of picture points in the run. The GLRLM matrix describes each neighborhood local texture. [4, 1, 8, 7]. In the following computations the nomenclature is as follows:

$g(i,j)$ is the element in the cell (i,j) of the normalized GLRLM

i is related the pixel intensity and j to the length of the run

`itk::itkRunLengthTextureFeaturesImageFilter` computes the following features:

Short run emphasis (SRE) measures the distribution of short runs. SRE is expected large for fine textures.

$$f_1 = \frac{\sum_{i,j}\frac{g(i,j)}{j^2}}{\sum_{i,j} g(i,j)} \tag{9}$$

Long run emphasis (LRE) is a feature that measures distribution of long runs. LRE is expected large for coarse structural textures.

$$f_2 = \frac{\sum_{i,j} g(i,j) j^2}{\sum_{i,j} g(i,j)} \tag{10}$$

Grey level non-uniformity (GLN) measures the similarity of grey-level values through out the texture. The GLN is expected small if the grey-level values are alike throughout the whole texture.

$$f_3 = \frac{\sum_i(\sum_j g(i,j))^2}{\sum_{i,j} g(i,j)} \tag{11}$$

Run length non-uniformity (RLN) is a feature that measures the similarity of the length of runs through out the image. The RLN is expected small if the run lengths are alike through out the image.

$$f_4 = \frac{\sum_j(\sum_i g(i,j))^2}{\sum_{i,j} g(i,j)} \tag{12}$$

Low grey level run emphasis (LGRE) is orthogonal to SRE, and the value of the feature increases when the texture is dominated by many runs of low gray value.

$$f_5 = \frac{\sum_{i,j}\frac{g(i,j)}{i^2}}{\sum_{i,j} g(i,j)} \tag{13}$$

High grey level run emphasis (HGRE) is orthogonal to LRE, and the metric increases when the texture is dominated by many runs of high gray value.

$$f_6 = \frac{\sum_{i,j} g(i,j) i^2}{\sum_{i,j} g(i,j)} \tag{14}$$

Short run low grey level emphasis (SRLGE) is a diagonal measurement that combines SRE and LGRE. The metric increases when the texture is dominated by many short runs of low gray value.

$$f_7 = \frac{\sum_{i,j} \frac{g(i,j)}{i^2 j^2}}{\sum_{i,j} g(i,j)} \tag{15}$$

Short run high grey level emphasis (SRHGE) is orthogonal to SRLGE and LRHGE and increases when the texture is dominated by short runs with high intensity levels.

$$f_8 = \frac{\sum_{i,j} \frac{g(i,j) i^2}{j^2}}{\sum_{i,j} g(i,j)} \tag{16}$$

Long run low grey level emphasis (LRLGE) is complementary to SRHGE, it increases when the texture is dominated by long runs that have low gray levels.

$$f_9 = \frac{\sum_{i,j} \frac{g(i,j) j^2}{i^2}}{\sum_{i,j} g(i,j)} \tag{17}$$

Long run high grey level emphasis (LRHGE) is the complementary metric to SRLGE and increases with a combination of long, high-gray value runs.

$$f_{10} = \frac{\sum_{i,j} g(i,j) i^2 j^2}{\sum_{i,j} g(i,j)} \tag{18}$$

3 Filters usage

3.1 itkCoocurrenceTextureFeaturesImageFilter

For each pixel of the input image, the itkCoocurrenceTextureFeaturesImageFilter will compute a serie of 8 co-occurrence texture features which will be contained in a vector. That way the output of the filter is a N-D image where each pixel will contain a vector of 8 scalars. Each texture map can be extracted from the output image afterward thanks to `itk::NthElementImageAdaptor`. By default the texture features are computed for each spatial direction and averaged afterward to provide rotationally invariant texture descriptors.

Template Parameters (if used in C++):

- The input image type: a N-Dimensional image where the pixel type MUST be integer.
- The output image type: a N-Dimensional image where the pixel type MUST be a vector of floating points or an ImageVector.

Inputs and parameters:

- An input image
- A mask defining the region over which texture features will be calculated. (Optional)
- The pixel value that defines the ”inside” of the mask. (Optional, defaults to 1 if a mask is set.)
- The number of intensity bins. (Optional, defaults to 256.)
- The set of directions (offsets) to average across. (Optional, defaults to (-1, 0), (-1, -1), (0, -1), (1, -1) for 2D images and scales analogously for ND images.)
- The pixel intensity range over which the features will be calculated. (Optional, defaults to the full dynamic range of the pixel type.)
- The size of the neighborhood radius. (Optional, defaults to 2.)

3.2 itkRunLengthTextureFeaturesImageFilter

For each pixel of the input image, the itkRunLengthTextureFeaturesImageFilter will compute a serie of 10 run length texture features which will be contain in a vector. That way the output of the filter is a N-D image where each pixel will contain a vector of 10 scalars. Each texture map can be extracted from the output image afterward thanks to `itk::NthElementImageAdaptor`. By default the texture features are computed for each spatial direction and averaged afterward.

Template Parameters:

- The input image type: a N dimensional image where the pixel type MUST be integer.
- The output image type: a N dimensional image where the pixel type MUST be a vector of floating points or an ImageVector.

Inputs and parameters:

- An input image
- A mask defining the region over which texture features will be calculated. (Optional)
- The pixel value that defines the ”inside” of the mask. (Optional, defaults to 1 if a mask is set.)
- The number of intensity bins. (Optional, defaults to 256.)
- The set of directions (offsets) to average across. (Optional, defaults to (-1, 0), (-1, -1), (0, -1), (1, -1) for 2D images and scales analogously for ND images.)
- The pixel intensity range over which the features will be calculated. (Optional, defaults to the full dynamic range of the pixel type.)
- The distance range over which the features will be calculated. (Optional, defaults to the full dynamic range of double type.)
- The size of the neighborhood radius. (Optional, defaults to 2.)

3.3 Recommendations

Using the itkTextureFeature's filters with the default settings will lead, in all likelihood, to meaningless results. In addition, those results might be really time consuming to compute.

To obtain significant results, most of the parameters need to be carefully chosen depending on the scale of the texture and resolution of the input data, the other parameters and the significant information that need to be revealed by the output. For example the pixel intensity range should be adapted to the actual range of the input data in order to obtain a larger amplitude in the output feature maps. The radius (of the neighborhood) should be chosen depending on the size of the anomaly/object that needs to be detected in the input image. The distance range should correspond, in most cases, to the longer run length possible in the neighborhood (it depend of the spacing in the input image, the radius of the neighborhood and the offset directions).The choices of the number of bins should be adapted to the intensity/run length variation that want to be observed as well as the intensity/distance ranges specified. Finally the offset direction can be adapted if the anomaly/object that needs to be detected is specific to one known direction of the input image.

The usage of a Region Of Interest (ROI) mask is strongly advised, it will reduce the computation time by avoiding computing features for the noisy/background parts of the image.

In addition to the settings, particular attention should be payed to the input data. Please consider cropping the input are to contain only areas that will be interesting for the analysis. This will both help improving the computation time, thanks to a better distribution of the threaded regions and avoiding memory problems due to too large output data (considering that the output data is 8 or 10 times bigger than the input data).

The memory problem due to too large output data can also be solved by separating the output image containing all the feature maps into several images containing one feature map each thanks to the itk class `itk::VectorIndexSelectionCastImageFilter`.

3.4 Python wheels

Python wheels allow to easily install itkTextureFeatures and all its dependencies in order to have this texture filters ready to use in python code.They have been generated for the three main operating systems (Mac, Linux and Windows) and three versions of python (2.7, 3.5 and 3.6).

To install the python wheels use the following command-line: $ pip install itk_textureFextures

4 Practical examples

4.1 C++

```
#include "itkRunLengthTextureFeaturesImageFilter.h"

#include "itkImage.h"
#include "itkVector.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkNeighborhood.h"

int main(int argc, char * argv[])
{
    if( argc != 10 )
    {
    std::cerr << "Usage: "<< std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFilePath> <MaskFilePath> <OutputFilePath> ";
    std::cerr << " <NumberOfBinsPerAxis> <PixelValueMin> ";
    std::cerr << " <PixelValueMax>  <DistanceValueMin> " ;
    std::cerr << " <DistanceValueMax> <NeighborhoodRadius> ";
    std::cerr << std::endl;
    return EXIT_FAILURE;
    }

    // Setup types
    typedef itk::Image< int, 3 >                        InputImageType;
    typedef itk::Image< itk::Vector< float, 10 > , 3 >  OutputImageType;
    typedef itk::ImageFileReader< InputImageType >      readerType;
    typedef itk::Neighborhood<typename InputImageType::PixelType,
      InputImageType::ImageDimension> NeighborhoodType;
    NeighborhoodType neighborhood;

    // Create and setup a reader
    readerType::Pointer reader = readerType::New();
    reader->SetFileName( argv[1] );

    // Create and setup a maskReader
    readerType::Pointer maskReader = readerType::New();
    maskReader->SetFileName( argv[2] );

    // Apply the filter
    typedef itk::Statistics::RunLengthTextureFeaturesImageFilter
                            < InputImageType, OutputImageType > FilterType;
    FilterType::Pointer filter = FilterType::New();
    filter->SetInput(reader->GetOutput());
    filter->SetMaskImage(maskReader->GetOutput());
    filter->SetNumberOfBinsPerAxis(std::atoi(argv[4]));
    filter->SetPixelValueMinMax(std::atof(argv[5]),std::atof(argv[6]));
    filter->SetDistanceValueMinMax(std::atof(argv[7]),std::atof(argv[8]));
    neighborhood.SetRadius( std::atoi(argv[9]) );
    filter->SetNeighborhoodRadius(neighborhood.GetRadius());

    // Create and setup a writter
    typedef  itk::ImageFileWriter< OutputImageType  > WriterType;
    WriterType::Pointer writer = WriterType::New();
    std::string outputFilename = argv[3];
```

```
    writer->SetFileName(outputFilename.c_str());
    writer->SetInput(filter->GetOutput());
    writer->Update();

  return EXIT_SUCCESS;
}
```

4.2 Python

```
import itk, sys

if len(sys.argv) != 7:
    print("Usage: " + sys.argv[0] + " <inputImagePath> <maskImagePath>"
                                   " <NumberOfBinsPerAxis> <PixelValueMin> "
                                   "<PixelValueMax> <NeighborhoodRadius>")
    sys.exit(1)

im = itk.imread(sys.argv[1])
mask = itk.imread(sys.argv[2])

filtr = itk.CoocurrenceTextureFeaturesImageFilter.New(im)
filtr.SetMaskImage(mask)
filtr.SetNumberOfBinsPerAxis(int(sys.argv[3]))
filtr.SetPixelValueMinMax(int(sys.argv[4]), int(sys.argv[5]))
filtr.SetNeighborhoodRadius([int(sys.argv[6]),int(sys.argv[6]),int(sys.argv[6])])

result = filtr.GetOutput()

itk.imwrite(result, "result.nrrd")
```

5 Results

We present concrete use case scenarios of the different filters of itkTextureFilters in this section. We used itkTextureFilters to characterize subchondral bone texture in temporomandibular joint (TMJ) Osteoarthritis (OA). To date, there is no single sign, symptom, or test that can clearly diagnose early stages of TMJ OA. However, it has been observed that changes in the subchondral bone occur in very early stages of this disease involving subchondral bone structural changes (texture) in the subchondral bone (i.e. bone marrow).

The different tools presented in this document aid detecting and highlighting those texture variations in order to help clinicians to detect TMJ OA earlier.

In the test case (figure 1), the lower part of the condyle is healthy (normal bone trabeculae density) whereas the upper part is characteristic of a TMJ OA case (low bone trabeculae density).

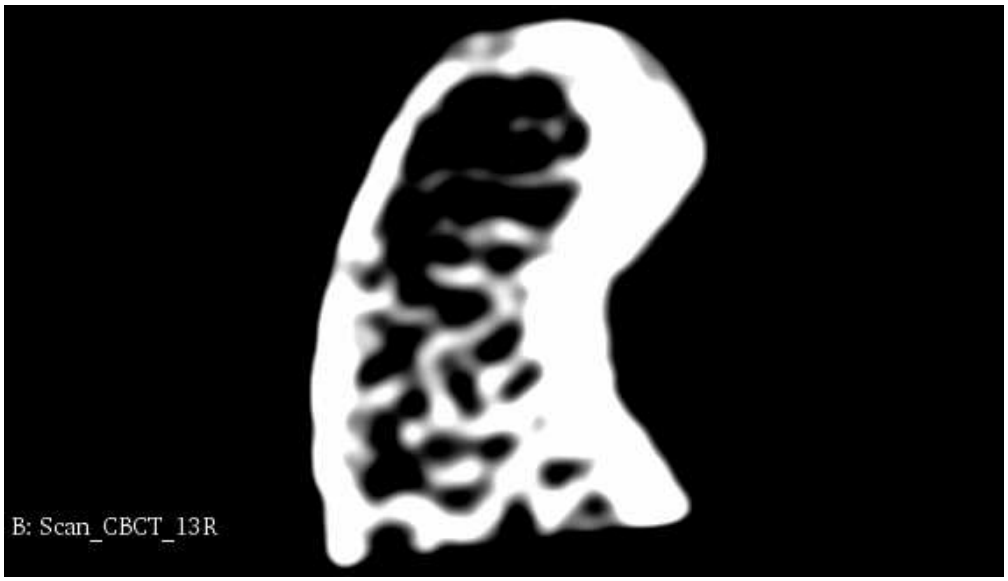

Figure 1: CBCT of the test condyle: this condyle suffers of a lack of trabecula in the upper part

5.1 itkCoocurrenceTextureFeaturesImageFilter

The results exposed in this part were obtained by specifying the following parameters (the default parameters were used for the other ones):

- Input data: Scan_CBCT_13R.nrrd
- Input mask: SegmC_CBCT_13R.nrrd
- Number of intensity bins: 10
- Pixel intensity range: [0; 4200]
- Neighborhood Radius: 6

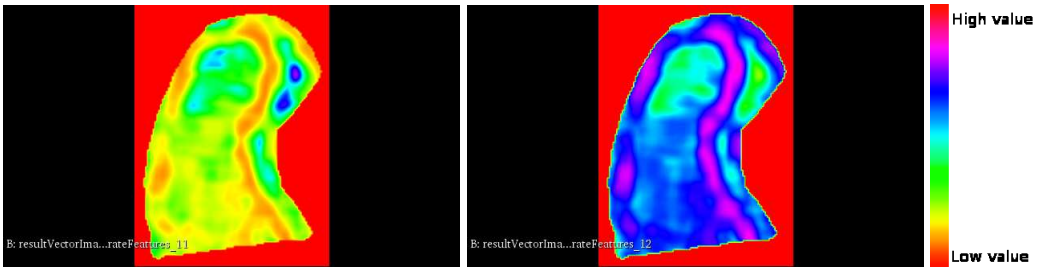

Figure 2: Texture's energy (left) and texture's entropy (right)

Figure 3: Texture's correlation (left), texture's inverse difference moment (center) and texture's inertia (right)

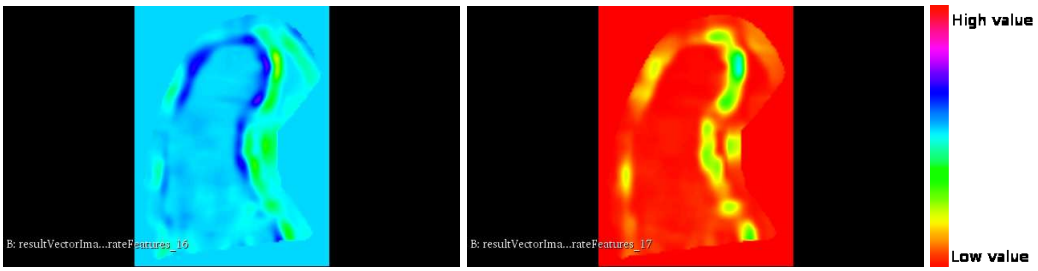

Figure 4: Texture's cluster shade (left) and texture's cluster prominence (right)

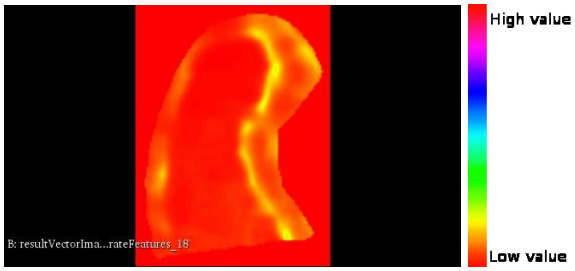

Figure 5: Texture's Haralick correlation

It seems that, the energy (figure 2), the entropy (figure 2), the correlation (figure 3), the inverse difference moment (figure 3), and the inertia (figure 3) are discriminating the healthy part of the bone trabecula from

the unhealthy part of the bone trabecula. The detection of a significant variation in several of those feature maps could be interpreted in an early TMJ OA. In this particular case, the cluster shade feature map (figure 4), cluster prominence feature map (figure 4) and hahlick correlation feature map (figure 5) don't seem to discriminate different texture types within the TMJ condyle.

5.2 itkRunLengthTextureFeaturesImageFilter

The results exposed in this part were obtained by specifying the following parameters (the default parameters were used for the other ones):

- Input data: Scan_CBCT_13R.nrrd
- Input mask: SegmC_CBCT_13R.nrrd
- Number of intensity bins: 10
- Pixel intensity range: [0; 4200]
- Distance range: [0; 1.25]
- Neighborhood Radius: 4

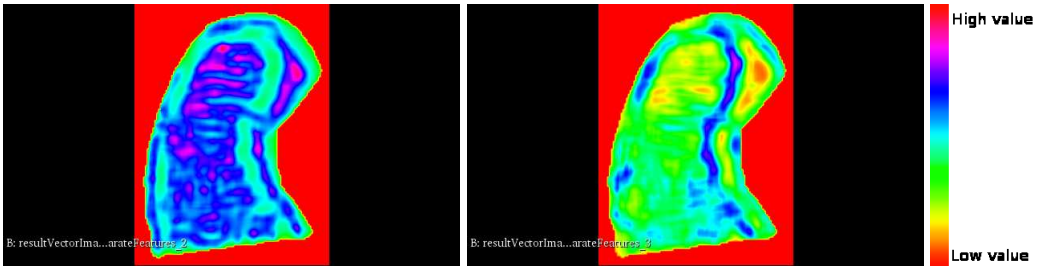

Figure 6: Texture's grey level non uniformity (left) and texture's run length non uniformity (right)

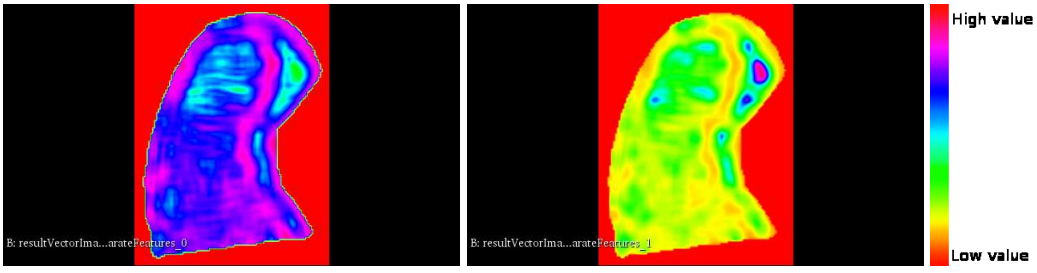

Figure 7: Texture's short run emphasis (left) and texture's long run emphasis (right)

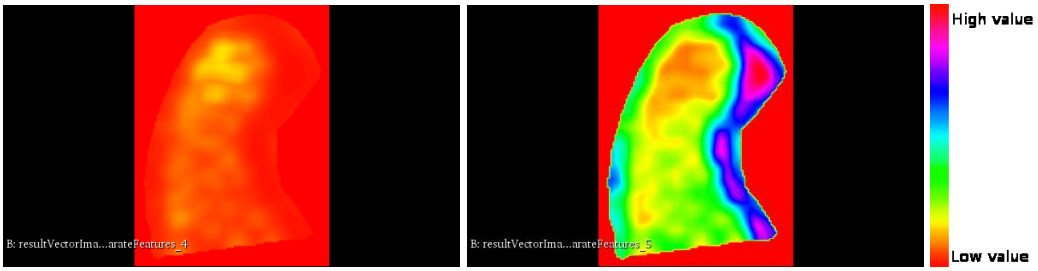

Figure 8: Texture's low grey level run emphasis (left) and texture's high grey level run emphasis (right)

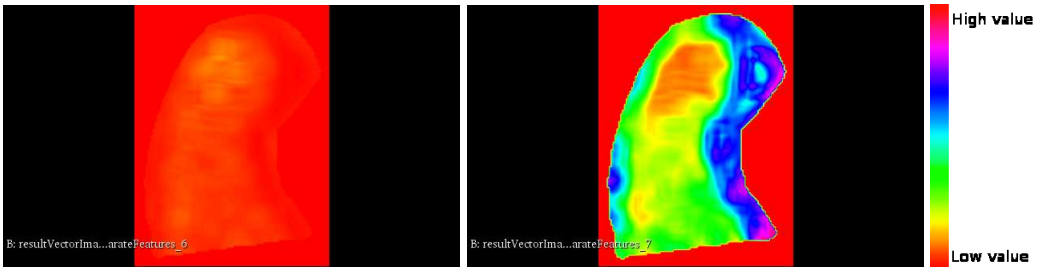

Figure 9: Texture's short run low grey level emphasis (left) and texture's short run high grey level run emphasis (right)

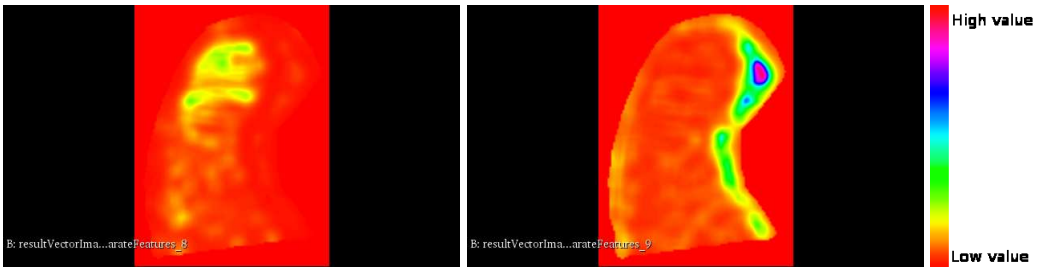

Figure 10: Texture's long run low grey level emphasis (left) and texture's long run high grey level emphasis (right)

Similarly to itkCoocurrenceTextureFeaturesImageFilter, some output feature maps of itkRunLengthTextureFeaturesImageFilter characterize bone texture and provide discriminant values of low and high density bone texture. Run length non uniformity (figure 6), short run emphasis (figure 7), long run emphasis (figure 7), high grey level run emphasis (figure 8), short run high grey level run emphasis (figure 9) and long run low grey level emphasis (figure 10) seem to have discriminant properties and differentiate healthy and pathological bone trabecula. Here again some feature maps are not helpful in this particular case.

6 Conclusion

This document presented a new fast and efficient tool to compute textural features in a N-Dimensional image. These features are used to describe and detect variations in the image's texture. A lot of the described features are correlated to each other, so probably not using all of them at the same time is important. The variety of features available allow to detect a large type of variations, and using any combination of them might help texture characterization and discrimination. This method is currently used in a study aiming to create a new method to detect TMJ OA at early stages using subchondral bone texture as a biomarker.

Acknowledgements

This work was supported by the National Institute of Health (NIH) National Institute for Dental and Craniofacial Research (NIDCR) grant R21DE025306 (Textural Biomarkers of Arthritis for the Subchondral Bone in the Temporomandibular Joint), NIDCR grant R01DE024450 (Quantification of 3D bony Changes in Temporomandibular Joint Osteoarthritis) and National Institute of Biomedical Imaging and Bioengineering (NIBIB) grant R01EB021391 (Shape Analysis Toolbox for Medical Image Computing Projects).

We would like to thank Dr. Larry Wolford from the Baylor University Medical Center for kindly providing the bone specimens from which we obtained the scans used in the paper. We would like to thank Drs. Lucia Cevidanes, Erika Benavides and Antonio Ruellas at the University of Michigan School of Dentistry as well, for generating the CBCT scans that were processed with the filters presented in the paper.

We are also grateful for the support received from the ITK community.

References

[1] A. Chu, C.M. Sehgal, and J. F. Greenleaf. Use of gray value distribution of run length for texture analysis. *Pattern Recognition Letters*, 11(6):415–419, 1979. 2.2

[2] R.W. Conners and C.A. Harlow. A theoretical comaprison of texture algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2:204–222, 1980. 2.1

[3] R.W. Conners, M.M. Trivedi, and C.A. Harlow. Segmentation of a high-resolution urban scene using texture operators. *Computer Vision, Graphics and Image Processing*, 25:273–310, 1984. 2.1

[4] M.M. Galloway. Texture analysis using gray level run lengths. *Computer Graphics and Image Processing*, 4(2):172–179, 1975. 2.2

[5] R.M. Haralick. Statistical and structural approaches to texture. *Proceedings of the IEEE*, 67:786–804, 1979. 2.1

[6] R.M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE Transactions on Medical Imaging*, 3(6):610–620, 1973. 2.1

[7] X. Tang. Texture information in run-length matrices. *IEEE Transactions on Image Processing*, 7(11):1602–1609, 1998. 2.2

[8] D. Xu, A. Kurani, J. Furst, and D. Raicu. Run length encoding for volumetric texture. *International Conference on Visualization, Imaging and Image Processing (VIIP)*, pages 452–458, 2004. 2.2