Radiomics Calculator

Generated by Doxygen 1.8.13

Contents

Chapter 1

RaCaT

Welcome to the documentation of RaCaT!

RaCaT can be used with any kind of images (like CT, PET and MRI) in dicom, ecat, nrrd or nifti format. Masks marking the region of interest can be imported as binary masks (also in dicom, ecat, nrrd or nifti format) or as rt-struct.
RaCat comes as an executable and does not require any further installations. As input it needs a configuration file where you can set the preprocessing steps (like discretization, resegmentation etc) you want to use. For more explanation see the next sections.

1.1 Getting started

In order to use the Radiomics Calculator, the Radiomics.exe has to be downloaded. In the folder ExampleFiles, you can find examples of the configuration files and other files you can give to the tool (optional). Every file is explained in detail in the following section.

- Download and adapt config.ini
  In the config.ini file you can set several preprocessing parameters. You can set e.g. if you want to use a discretization step (or not), if your image is a PET or CT image, if you want to apply resegmentation etc. More information about the config file can be found in the section "Setting up the config.ini file".

- Optional: Download and adapt featureSelection.ini
  RaCaT calculates a large number of feature values. The features are ordered in several groups, e.g. Statistical features, Grey-level-cooccurrence features etc. If you want to calculate only certain features, you can say this in the featureSelection.ini file. This file is not required to make RaCaT work. So you only have to change and adapt it if you want to calculate only certain features. Then you also have to give the path where you saved your featureSelection file as parameter to the executable. In the ExampleFiles folder, there are some examples for feature selection files and how you can call the tool if you want to include a feature selection file. If no featureSelection file is given, all features are calculated.

- Only for PET images: Download and adapt patientInfo.ini
  If your image is a PET image, RaCaT needs the patient information so it can convert the intensity values in the image from Bq/ml to SUV. For this, you can download the patientInfo.ini file and fill in your patient parameters like weight etc.

- Call the executable After adapting these files, you can call the executable. As parameters it needs:

  - path to the ini-file
  - path to the image
  - path to the mask
  - path to folder where you want to save the images (if the folder does not exist, the folder is created)
    A lot of examples on how to call the executable for PET/CT images, with or without featureSelection.ini can be found in the example folder. IF the mask is a binary image in dicom/nifti/nrrd format, the executable can be called in the following way:

```
/path/to/.exe --ini /path/to/iniFile --img /path/to/prj/image file  --voi /path/to/voi(mask) file  --out
       /path/to/outputfolder
```

    The output folder will be created automatically. If the folder already exists, a new folder will be created which has the name of the required output folder including the time stamp of the calculation. If the mask is a RT-struct, –voi has to be replaced by –rts:

```
/path/to/.exe --ini /path/to/iniFile --img /path/to/prj/image file  --rts /path/to/rt struct  --out
       /path/to/outputfolder
```

- Help With –h, a help is called, that gives an example of how the executable should be called.
- Example data Example files for config, feature selection, and patient information are stored in the folder 'ExampleFiles'. Also batch-files with examples how to call the executable can be found in this folder.

Chapter 2

Build Radiomics Toolbox from source

In order to build the Radiomics Toolbox from source, you have to follow the following steps:

1. Clone Radiomics repository Download the files of this repository.
2. Get CMake CMake can be downloaded here: CMAKE
3. Get ITK
   (a) Windows Here we only describe the installation using Microsoft Visual Studio. For all other compilers, check: ITK getting started Windows
      - Launch CMake GUI
      - Go to field "Where is the source code:", click "Browse Source..." and navigate to where you cloned the repository with Git.
      - Go to "Where to build the binaries:", select "Browse Build..." and select a place to build the ITK library. It should NOT be the same directory as the one where you cloned the repository.
      - Click "Configure", and then specify "Visual Studio x" as the generator for this project.
      - Choose your build options
      - Click "Generate".
      - Open Visual Studio x and open the ALL_BUILD project that is in the folder where you built the ITK project
      - Click F5 (run solution)

   (b) Linux
      - Download the ITK source code: ITK source code
      - unpack tarball: sudo tar xvzf InsightToolkit-3.14.0.tar.gz
      - Create a directory, where ITK should be built, e.g:
        `sudo mkdir /usr/local/itk/InsightToolkit-3.14.0/ITK-build`
      - Go to this directory and run ccmake:
        `sudo ccmake -DITK_USE_REVIEW=ON`
      - Press "c" to configure and "g" to generate
      - Then you can run ITK:
        `sudo make`
      - Then you can install ITK:
        `sudo make install`
4. Get boost The Radiomics Toolbox needs also the boost library. Download the source code from here: Boost and unpack the folder. We have to install some boost libraries:
      - Go to the folder, where you unpacked the boost libraries

- Type following command in command line:

  `./bootstrap.sh`

- Type the command:

  `./b2 install` Boost is installed.

5. Use CMakeList to configure, generate and compile Radiomics.exe In order to run the radiomics code with the libraries, the following CMakeLists.txt file is required: The /PATH/TO elements have to be replaced by the corresponding paths on your computer.

```
cmake_minimum_required(VERSION 2.8)
project(Radiomics)
SET (BOOST_ROOT /PATH/TO/BOOST)
SET (BOOST_LIBRARYDIR "/PATH/TO/BOOST/stage/lib")
SET (BOOST_MIN_VERSION "1.55.0")
set (Boost_NO_BOOST_CMAKE ON)
FIND_PACKAGE(Boost ${BOOST_MIN_VERSION} REQUIRED)
if (NOT Boost_FOUND)
message(FATAL_ERROR "Fatal error: Boost (version >= 1.55) required.")
else()
message(STATUS "Setting up BOOST")
message(STATUS " Library  - ${Boost_LIBRARY_DIRS}")
include_directories(${Boost_INCLUDE_DIRS})
link_directories(${Boost_LIBRARY_DIRS})
endif (NOT Boost_FOUND)
find_package(ITK REQUIRED)
include(${ITK_USE_FILE})
add_executable(Radiomics MACOSX_BUNDLE /PATH/TO/radiomics-master/main.cpp)
target_link_libraries(Radiomics
${Boost_LIBRARIES} ${Glue}  ${VTK_LIBRARIES} ${ITK_LIBRARIES})
```

   Open the CMAKE GUI.

6. Compile the Radiomics project with Microsoft Visual Studio

7. Now you can use the Radiomics.exe as described

Chapter 3

Setting up the config.ini file

In the config.ini file you can set the preprocessing steps you want to use.

- Smoothing:
  If you want to use additional smoothing to the image, you can set a Gaussian kernel to the desired full-width-at-half-maximum. Smoothing is applied as soon as the Smoothing kernel is not equal to 0.

- Threshold (in %) for including voxels in the VOI: Masks can contain different values. A mask can contain only 1s, values from 1-100, or other ranges. You can determine which voxels will be included in the final mask by setting a threshold. The program determines the maximum value inside the mask and includes all values in the final mask which have a value higher than this threshold from the maximum values. The recommended value is 0.5.
  The value is especially important when you apply resampling to the image and the mask.

- Discretization method:
  Before textural features are calculated, the image is normally discretized. You can choose between two different methods for discretization:
  - use fixed bin width and/or
  - use fixed nummer of bins
    You can set the bin width and the number of bins you want to use.

- Interpolation method:
  If you want to up- or downsample the image, you can say it in the part "interpolation method". You can choose between up- or downsampling or if you want to sample the image to cubic voxels of 2 mm voxel size.
  For interpolation, trilinear interpolation is used.

- Resegmentation:
  If you want to resegment the image region, you can state that here. I.e. if you want to exclude values above/below a maximum/minimum value or if you want to exclude outliers from the segmentation.
  Herefore, you have to set ReSegmentImage to 1 and set the minimum and maximum value you want to include.
  For PET images, the minimum and maximum values have to be in kBq/ml.

- Image properties:
  Here you can set the image type you are using. In general, the program is only interested if you have a PET image or not:
  If the image is a PET image, a patientInfo.ini file is required. In this file, you can set values typical for PET images.
  If another image type is set, the patientInfo.ini file is not required and will not be read in.

- NGLDMParameters:
  The neighbourhood grey level dependence matrix (NGLDM) captures the texture of the matrix. It compares the intensity values of neighboring voxels with the intensity value of a center voxel. If the difference between these intensity values is smaller than a threshold (coarseness parameter), the voxels are defined to be independent. The default coarseness parameter is 0, but you can change it to other values in this section.
  Furthermore, you can define the size of a neighborhood: dist is the distance of a voxel to the center voxel. All voxels with a distance smaller or equal to this distance are regarded as part of the neighborhood. The default value is 1.

- NGTDM parameters:
  You can set the size (distance to center voxel) of a neighborhood. The default value is also 1.

- Distance weight properties:
  The contents of GLCM, GLRLM and NGTDM matrices can be updated by the distance between voxels. Herefore, different definitions of the distance can be used. Possible distances are:

  - Chebyshev (default)
  - Manhattan
  - Euclidean norm
    If another distance is set, the program gives an error.

- Extended Emphasis Features:
  In the GLRLM, GLSZM, GLDZM and NGLDM matrices, you can set a particular emphasis on part of these matrices. In the formula of the feature calculations, the emphasis can be set by different powers. The desired powers can be set here.
  If the extended emphasis features should be calculated, the CalculateExtendedEmph value should be set to 1.

- Output information:
  Here, you can set the output format. You can choose between csv or ontology output. If you want to have a csv file as output, you can decide if you want to have it as one csv file or one csv file per feature group.
  The class ConfigFile has as attributes exactly the attributes which can be set in the config-file.

Chapter 4

Setting up the patientInfo.ini file

The patientInfo.ini file is only required if a PET image is used as input. You can change the following parameters:

- Patient weight
  The patient weight is needed in order to calculate the SUV values.
  The SUV values are only calculated if the parameter UseSUV is set to 1.
  If the scaling parameter is set to a value **not** equal to 0, the weight is not needed.
- Patient height
  The patient height is needed in order to calculate the SUL values.
  The SUL values are only calculated if the parameter UseSUL is set to 1.
  If the scaling parameter is set to a value **not** equal to 0, the height is not needed.
- ActivityMBq
  The injected dose (in MBq) at scan **START**.
  Please note that no decay correction is done by the program. Therefore the dose has to be the one at scan start.
  If the scaling parameter is set to a value **not** equal to 0, the activity is ignored.
- Gender
  he gender is needed in order to calculate the SUL values.
  Possible Values are **M** or **F**. All other values are ignored.
  The SUL values are only calculated if the parameter UseSUL is set to 1.
  If the scaling parameter is set to a value **not** equal to 0, the height is not needed.
- UseSUV
  The value has to be set to 1, if the image should be converted in SUV.
- UseSUL
  The value has to be set to 1, if the image should be converted to SUL.
- ScalingFactor
  If the scaling factor is set to a value **not** equal to 0, all other values will be ignored and the image will be scaled by this factor.
  UseSUV has to be set to 1.

Chapter 5

Feature calculation and setting up the featureSelection.ini file

In the file featureSelection.ini, the user can set which features should be calculated.
The feature selection file is splitted in several paragraphs: For every feature group one. Setting the value in one paragraph to 0 will exclude this feature group from the calculation step.
E.g. if in the section [StatisticalFeatures] the value `CalculateStatFeat = 0`, the statistical features are not calculated.
The following feature groups are available. For exact explanations of the implementation, check the class documentation. For the feature definitions, check Zwanenburg
Features that do not require interpolation:

- Morphological features
  Features describing the tumor shape like volume, surface etc.

- Local intensity features
  Features describing local or global intensity values like SUVpeak.

- Statistical features First order features like the mean, maximum or minimum intensity value of the VOI.

  Textural features requiring interpolation. All these feature capture heterogeneity information about the VOI.

- Grey Level Co-occurence based features
  For the Grey Level Co-occurence features, a matrix is created that captures information about the neighboring voxels in the VOI.
  The number of times a certain pair of neighboring voxels occur in the VOI is stored in this matrix. In this way, the matrix measures if there are a lot of neighboring voxels with high intensity differences and captures information about the heterogeneity/homogeneity of the VOI.
  For the grey level co-occurence features, five different ways of computing exist.

- Grey Level Run length based features
  In order to calculate the GLR based features, a matrix is calculated that contains the information how often a certain grey level occurs in a row in the VOI. The run length is captured for different angles.
  In this way, also the GLR features capture heterogeneity information.
  Also for this feature group, five different ways of creating the matrix exist. All five are implemented here.

- Grey Level Size Zone based features
  The GLSZ features capture information about how many connected voxels with the same intensity values can be found in the VOI.
  For this feature group, three ways of creating the matrix exist.

- Grey Level Distance Zone based features
  The GLDZ based features also contain information about how many connected voxels with the same intensity values can be found in the VOI.
  On top of that, it also stores the information how far away the minimum distance of this connected zones is from the border of the VOI.
  Also for this feature group, three different ways of building the matrix exist.

- Neighborhood Grey Tone Difference based features
  The NGTD contains the sum of grey level differences of voxels with the same grey level and the average grey level of neighbouring voxels.
  The user can define the size of the neighborhood in the config.ini file. The default value is 1.

- Neighborhood Grey Level Dependence based features
  For the NGLD matrices, for every voxel of the VOI a neighbourhood is defined as the voxels located within a distance d around this voxel. If the intensity differences of this voxel and the neighboring voxels are smaller than a certain value (coarseness parameter), the voxels are regarded as independent. The NGLD matrices contain this information.
  The user can set the size of the neighborhood, as well as the coarseness parameter in the config.ini file.
  This matrices contain information about the coarseness of the VOI.

Chapter 6

README

In order to make the RadiomicsToolbox run, you have to follow the following steps:

1. Clone Radiomics repository

Download the files of this repository.

1. Get CMake CMake can be downloaded here: https://cmake.org/install/
2. Get ITK 3.1 Windows Here we only describe the installation using Microsoft Visual Studio. For all other compilers, check: https://itk.org/Wiki/ITK/Getting_Started/Build/Windows
    - Launch CMake GUI
    - Go to field "Where is the source code:", click "Browse Source..." and navigate to where you cloned the repository with Git.
    - Go to "Where to build the binaries:", select "Browse Build..." and select a place to build the ITK library. It should NOT be the same directory as the one where you cloned the repository.
    - Click "Configure", and then specify "Visual Studio x" as the generator for this project.
    - Choose your build options
    - Click "Generate".
    - Open Visual Studio x and open the ALL_BUILD project that is in the folder where you built the ITK project
    - Click F5 (run solution) 3.2 Linux
    - Download the ITK source code: https://itk.org/ITK/resources/software.html
    - unpack tarball: sudo tar xvzf InsightToolkit-3.14.0.tar.gz
    - Create a directory, where ITK should be built, e.g: sudo mkdir /usr/local/itk/InsightToolkit-3.14.0/ITK-build
    - Go to this directory and run ccmake: sudo ccmake -DITK_USE_REVIEW=ON
    - Press "c" to configure and "g" to generate
    - Then you can run ITK: sudo make
    - Then you can install ITK: sudo make install
3. Get boost The Radiomics Toolbox needs also the boost library. Download the source code from here: http://www.boost.org/users/history/version_1_61_0.html and unpack the folder. We have to install some boost libraries:
    - Go to the folder, where you unpacked the boost libraries

- Type following command in command line: ./bootstrap.sh
- Type the command: ./b2 install Boost is installed.

4. Use CMakeList to configure, generate and compile Radiomics.exe In order to run the radiomics code with the libraries, the following CMakeLists.txt file is required: The /PATH/TO elements have to be replaced by the corresponding paths on your computer. cmake_minimum_required(VERSION 2.8) project(↩ Radiomics) SET (BOOST_ROOT /PATH/TO/BOOST) SET (BOOST_LIBRARYDIR "/PATH/TO/BOOS↩ T/stage/lib") SET (BOOST_MIN_VERSION "1.55.0") set (Boost_NO_BOOST_CMAKE ON) FIND_PA↩ CKAGE(Boost ${BOOST_MIN_VERSION} REQUIRED) if (NOT Boost_FOUND) message(FATAL_ER↩ ROR "Fatal error: Boost (version >= 1.55) required.") else() message(STATUS "Setting up BOO↩ ST") message(STATUS " Library - ${Boost_LIBRARY_DIRS}") include_directories(${Boost_INCLUDE_↩ DIRS}) link_directories(${Boost_LIBRARY_DIRS}) endif (NOT Boost_FOUND) find_package(ITK REQ↩ UIRED) include(${ITK_USE_FILE}) add_executable(Radiomics MACOSX_BUNDLE /PATH/TO/radiomics-master/main.cpp) target_link_libraries(Radiomics ${Boost_LIBRARIES} ${Glue} ${VTK_LIBRARIES} ${IT↩ K_LIBRARIES})

   Open the CMAKE GUI.

5. Run Radiomics.exe

Chapter 7

Hierarchical Index

7.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 8

Class Index

8.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 9

File Index

9.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 10

Class Documentation

10.1 ConfigFile Class Reference

Public Member Functions

- config readIni (string iniFile)
- void getSmoothingKernel ()

  get smoothing kernel
- void getThreshold ()

  get threshold
- void getAccurateState (string accState)

  get information if we are working with prj file
- void getVoiState (string accState)

  get information with what kind of .voi we are working
- void getResegmentationState ()

  get information about resegmentation
- void getFeatureSelectionLocation (string featurePath)

  get the location of the featureSelection.ini
- void getDiscretizationInformation ()

  read the discretization information
- void getDiscretizationInformationIVH ()

  read discretization information IVH
- void getDistanceWeightProperties ()

  read distance weight properties
- void getExtendedEmphasisInformation ()

  get extended emphasis information
- void getNGLDMParameters ()

  get the NGLDM parameters
- void getNGTDMdistanceValue ()

  get the NGTDM distance value
- void getImageFolder (string imageName, string voiName)

  read the information about the folders where the images are saved
- void getOutputInformation (string output)

  read information about outputFolder location
- void getPETimageInformation (string imagePath, string patientInfoPath, ConfigFile config)

  read information necessary to convert image from kBq/ml to SUV

- void getInterpolation ()

  read information about interpolation
- void copyConfigFile (string outputFolder)

  copies the config file to the output folder
- void createOutputFolder (ConfigFile &config)
- void createConfigInfo (ConfigFile &config, string arguments[7])

  adjusts all values to the attributes in the ConfigFile

Public Attributes

- string **fileName**
- config **pt**
- float smoothingKernel

  float for smoothing kernel if !=0, image is smoothed
- float threshold

  threshold in %; all voxels which contain values higher than this percentage of the maximum mask value will be included in the mask
- int useAccurate

  integer which states if exe is called by accurate tool
- int voiFile

  integer which states if .voi is dicom, nii or else
- int useFixedBinWidth

  integer which states if we use fixed bin width
- float **binWidth**
- int useFixedNrBins

  integer which states if we use fixed number of bins
- int **nrBins**
- int **discretizeIVH**
- int **discretizeIVHSeparated**
- int **useFixedBinWidthIVH**
- int **binWidthIVH**
- int **useFixedNrBinsIVH**
- int **nrBinsIVH**
- int **useDownSampling**
- int **useUpSampling**
- int **useSampling2mm**
- int useReSegmentation

  integer which states if resegmentation is used
- int **excludeOutliers**
- float **minValueReSeg**
- float **maxValueReSeg**
- string **normGLCM**
- string **normGLRLM**
- string **normNGTDM**
- int extendedEmphasis

  shall the extended emphasis features be calculated?
- int **powerRow**
- int **powerCol**
- int distNGLDM

  parameters for NGLDM matrices
- int **coarsenessParam**
- int dist

distance defined by user for NGTDM matrices

- string **featureSelectionLocation**
- int **calculateAllFeatures** = 0
- string **patientInfoLocation**
- string imageName

  names of images and folders
- string **voiName**
- string **imageType**
- string outputFolder

  name of folder, where values are stored
- int getOneCSVFile

  integer which states which kind of output
- int **csvOutput**
- int **ontologyOutput**
- int useSUV

  parameters to calculate the SUV value in case we have a PET image; this values can be set in the patientInfo.ini file
- int **useSUL**
- float **patientWeight**
- float **patientHeight**
- int **malePatient**
- std::string **gender**
- float **initActivity**
- int **minAfterInjection**
- double correctionParam

10.1.1 Member Function Documentation

10.1.1.1 copyConfigFile()

```
void ConfigFile::copyConfigFile (
            string outputFolder )  [inline]
```

copies the config file to the output folder

The method copyConfigFile copies the config file in the output folder

10.1.1.2 createConfigInfo()

```
void ConfigFile::createConfigInfo (
            ConfigFile & config,
            string arguments[7] )  [inline]
```

adjusts all values to the attributes in the ConfigFile

The method createConfigInfo fills the class ConfigFile with all information provided in the config.ini file.
The method just executes all ini-methods.

10.1.1.3 createOutputFolder()

```
void ConfigFile::createOutputFolder (
            ConfigFile & config )  [inline]
```

The method createOutputFolder creates the outputFolder, if it does not already exists.
If it exists, a warning message is printed on the screen, that the data will be overwritten

10.1.1.4 getAccurateState()

```
void ConfigFile::getAccurateState (
            string accState )  [inline]
```

get information if we are working with prj file

The method getAccurateState sets the useAccurate value.
If we have a prj file, the value is set to 1.
If it is a dicom image, the value is set to 2.
Otherwise the value is 0.

10.1.1.5 getDiscretizationInformation()

```
void ConfigFile::getDiscretizationInformation ( )  [inline]
```

read the discretization information

The method getDiscretizationInformation reads the discretization information of the ini-file and sets the attributes of the class Config to the equivalent values

10.1.1.6 getDiscretizationInformationIVH()

```
void ConfigFile::getDiscretizationInformationIVH ( )  [inline]
```

read discretization information IVH

The method getDiscretizationInformationIVH reads the discretization information for the intensity volume histogram features and sets the attributes of the class Config to the equivalent values

10.1.1.7 getDistanceWeightProperties()

```
void ConfigFile::getDistanceWeightProperties ( )  [inline]
```

read distance weight properties

The method getDistanceWeightProperties reads the information concerning the distance weights.

10.1.1.8 getExtendedEmphasisInformation()

```
void ConfigFile::getExtendedEmphasisInformation ( )  [inline]
```

get extended emphasis information

The method getExtendedEmphasisInformation reads the information concerning the extended emphasis.

10.1.1.9 getFeatureSelectionLocation()

```
void ConfigFile::getFeatureSelectionLocation (
            string featPath )  [inline]
```

get the location of the featureSelection.ini

The method getFeatureSelectionLocation reads the location of the feature selection file

10.1.1.10 getImageFolder()

```
void ConfigFile::getImageFolder (
            string image,
            string voi )  [inline]
```

read the information about the folders where the images are saved

The method getImageFolder gets the image and voi path information the user provided in the command line. Furthermore it reads the image type.

10.1.1.11 getInterpolation()

```
void ConfigFile::getInterpolation ( )  [inline]
```

read information about interpolation

The method getInterpolation reads the interpolation information of the ini-file and sets the attributes of the class Config to the equivalent values

10.1.1.12 getOutputInformation()

```
void ConfigFile::getOutputInformation (
            string output )  [inline]
```

read information about outputFolder location

The method getOutputFolder gets the output folder path provided in the command line and reads the output information

10.1.1.13 getPETimageInformation()

```
void ConfigFile::getPETimageInformation (
            string imagePath,
            string patientInfoPath,
            ConfigFile con )  [inline]
```

read information necessary to convert image from kBq/ml to SUV

The method getPETimageInformation reads the pet image information of the patientInfo.ini-file and sets the attributes of the class Config to the equivalent values

10.1.1.14 getResegmentationState()

```
void ConfigFile::getResegmentationState ( )  [inline]
```

get information about resegmentation

The method getResegmentationState reads the provided resampling information.

10.1.1.15 getSmoothingKernel()

```
void ConfigFile::getSmoothingKernel ( )  [inline]
```

get smoothing kernel

The method getSmoothingKernel reads the value of the smoothing kernel.

10.1.1.16 getThreshold()

```
void ConfigFile::getThreshold ( )  [inline]
```

get threshold

The method getSmoothingKernel reads the value of the smoothing kernel.

10.1.1.17 getVoiState()

```
void ConfigFile::getVoiState (
            string voiState )  [inline]
```

get information with what kind of .voi we are working

The method getVoiState sets the voiFile value. It is 0 if we have a nifti or .voi file, if its a dicom image, the value is set to 2 and if it is a rt struct, the value is set to 3.

10.1.1.18 readIni()

```
config ConfigFile::readIni (
            string iniName )  [inline]
```

The method readIni reads the desired ini-file

Parameters

in		

10.1.2 Member Data Documentation

10.1.2.1 correctionParam

```
double ConfigFile::correctionParam
```

the correction parameter overwrites the SUV values set before (if bigger than 1) all image values will be divided by it, instead of dividing by calculated SUV

The documentation for this class was generated from the following file:

- readConfigFile.h

10.2 GLCMFeatures< T, R > Class Template Reference

```
#include <GLCMFeatures.h>
```

Inheritance diagram for GLCMFeatures< T, R >:

Public Member Functions

- void getXYDirections (int &directionX, int &directionY, int angle)
  getXYDirections
- std::vector< std::pair< T, T > > getNeighbours2D (boost::multi_array< T, R > inputMatrix, int depth, int directionX, int directionY)
- std::vector< std::pair< T, T > > getNeighbours3D (boost::multi_array< T, R > inputMatrix, int angle, int directionZ)
- void getDiagonalProbabilities (boost::multi_array< double, 2 > &glcMatrix)
  getDiagonalProbabilities
- void getCrossProbabilities (boost::multi_array< double, 2 > &glcMatrix)
  getCrossProbabilities
- void calculateMeanRowProb (boost::multi_array< double, 2 > glcMatrix)
  calculateMeanRowProb
- void calculateMeanColProb (boost::multi_array< double, 2 > glcMatrix)
  calculateMeanColProb
- void calculateRowProb (boost::multi_array< double, 2 > glcMatrix)
  calculateRowProb
- void calculateColProb (boost::multi_array< double, 2 > glcMatrix)
  calculateColProb
- void calculateJointMaximum (boost::multi_array< double, 2 > glcMatrix)

calculateJointMaximum

- void calculateJointAverage (boost::multi_array< double, 2 > glcMatrix)

  calculateJointAverage The joint average is the sum of joint probabilities. This sum is weighted by the grey level belonging to the probabilities.
- void calculateJointVariance (boost::multi_array< double, 2 > glcMatrix, T jointAvg)

  calculateJointVariance
- void calculateJointEntropy (boost::multi_array< double, 2 > glcMatrix)

  calculateJointEntropy
- void calculateDiffAverage ()

  calculateDiffAverage
- void calculateDiffVariance (T diffAverage)

  calculateDiffVariance
- void calculateDiffEntropy ()

  calculateDiffEntropy
- void calculateSumAverage ()

  calculateSumAverage
- void calculateSumVariance (T sumAverage)

  calculateSumVariance
- void calculateSumEntropy ()

  calculateSumEntropy The sum entropy is a measurement of the differences in the intensity value pairs
- void calculateAngSecMoment (boost::multi_array< double, 2 > glcMatrix)

  calculateAngSecMoment
- void calculateContrast (boost::multi_array< double, 2 > glcMatrix)

  calculateContrast
- void calculateDissimilarity (boost::multi_array< double, 2 > glcMatrix)

  calculateDissimilarity
- void calculateInverseDiff (boost::multi_array< double, 2 > glcMatrix)

  calculateInverseDiff
- void calculateInverseDiffNorm (boost::multi_array< double, 2 > glcMatrix, T inverseDiff)

  calculateInverseDiffNorm
- void calculateInverseDiffMom (boost::multi_array< double, 2 > glcMatrix)

  calculateInverseDiffMom
- void calculateInverseDiffMomNorm (boost::multi_array< double, 2 > glcMatrix)

  calculateInverseDiffMomNorm
- void calculateInverseVariance (boost::multi_array< double, 2 > glcMatrix)

  calculateInverseVariance
- void calculateCorrelation (boost::multi_array< double, 2 > glcMatrix)

  calculateCorrelation
- void calculateAutoCorrelation (boost::multi_array< double, 2 > glcMatrix)

  calculateAutoCorrelation
- void calculateClusterTendency (boost::multi_array< double, 2 > glcMatrix)

  calculateClusterTendency
- void calculateClusterShade (boost::multi_array< double, 2 > glcMatrix)

  calculateClusterShade
- void calculateClusterProminence (boost::multi_array< double, 2 > glcMatrix)

  calculateClusterProminence
- void calculateFirstMCorrelation (boost::multi_array< double, 2 > glcMatrix)

  calculateFirstMCorrelation
- void calculateSecondMCorrelation (boost::multi_array< double, 2 > glcMatrix)

  calculateSecondMCorrelation
- void **defineGLCMFeatures** (vector< string > &features)

Public Attributes

- vector< T > diagonalProbabilities

  vectors where the diagonal and cross probabilities are stored
- vector< T > **crossProbabilities**
- T jointMaximum

  define the different feature values of the GLCM matrix
- T **jointAverage**
- T **jointVariance**
- T **jointEntropy**
- T **diffAverage**
- T **diffVariance**
- T **diffEntropy**
- T **sumAverage**
- T **sumVariance**
- T **sumEntropy**
- T **angSecMoment**
- T **contrast**
- T **dissimilarity**
- T **inverseDiff**
- T **inverseDiffNorm**
- T **inverseDiffMom**
- T **inverseDiffMomNorm**
- T **inverseVar**
- T **meanRowProb**
- T **meanColProb**
- T **stdRowProb**
- T **stdColProb**
- T **autoCorrelation**
- T **correlation**
- T **clusterTendency**
- T **clusterShade**
- T **clusterProminence**
- T **firstMCorrelation**
- T **secondMCorrelation**

Private Attributes

- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**
- int **N_g**

10.2.1 Detailed Description

template<class T, size_t R = 3>
class GLCMFeatures< T, R >

A Grey-Level-Co-occurence-Matrix (GLCM) is used to calculate the spacial dependence of grey levels in an image. Before calculating the GLCM-matrix, the grey-values in the image have to be discretized. The GLC-matrix expresses how combinations of the discretized grey-levels of neighbor pixels/voxels are distributed along one of the image directions.

Two different approaches are available: The 2D- and the 3D-approach
In 3D for every voxel we have 26 direct neighbors and 13 unique directions.
In 2D we only look at the neighbors slice by slice, ignoring the connectivity between slices. In the 2D approach, for every pixel, there are 8 neighbors available.

In a GLC-matrix M_Δ the number of rows and columns is equal to the number of gray levels N_g in the region of interest. Δ is the direction of the neighbor: Four different unique directions are possible: 0, 45, 90 and 135 degrees. For all of these directions a GLC-matrix is calculated. Every GLC-matrix is calculated as: $M_\Delta = M_\delta + M_{-\delta}$.

Every matrix element m_{ij} of M_Δ is the relative frequency with which two pixels i and j occur within a given neighborhood (i is the intensity of one element and j is the intensity of its neighbor element).

After calculating the GLC-matrices, feature values can be calculated. Five different methods can be used to generate the feature values. For all of these methods, a separate class is defined:

- GLCMFeaturesWOMerge
- GLCMFeaturesWMerge
- GLCMFeaturesFullMerge
- GLCMFeatures3DWMerge
- GLCMFeatures3DWOMerge

All this classes inherit from the class GLCMFeatures. In this class all feature calculations are defined.

10.2.2 Member Function Documentation

10.2.2.1 calculateAngSecMoment()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateAngSecMoment (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateAngSecMoment

Parameters

matrix	GLC-matrix

The angular second moment is the same as the energy of the probability distribution

10.2.2.2 calculateAutoCorrelation()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateAutoCorrelation (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateAutoCorrelation

Parameters

matrix	GLC-matrix

AUto correlation measures the fineness or coarseness of the VOI.

10.2.2.3 calculateClusterProminence()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateClusterProminence (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateClusterProminence

Parameters

matrix	GLC-matrix

The cluster prominence gives also information about the skewness and asymmetry of the VOI (higher value: more assymetry)

10.2.2.4 calculateClusterShade()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateClusterShade (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateClusterShade

Parameters

matrix	GLC-matrix

The cluster shade measures the skewness and asymmetry of the VOI

10.2.2.5 calculateClusterTendency()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateClusterTendency (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateClusterTendency

Parameters

matrix	GLC-matrix

The cluster tendency gives information about the formation of voxels with similar grey values in groups

10.2.2.6 calculateColProb()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateColProb (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateColProb

Parameters

matrix	glcMatrix: GLC-matrix

Calculate the sum of the col probabilities and store them in vector sumProbCOls

formula see above

10.2.2.7 calculateContrast()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateContrast (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateContrast

Parameters

matrix	GLC-matrix

The contrast is a weighted sum of the elements of the GLC-matrix. The bigger the difference in the intensities of the neighbors the higher the weight

10.2.2.8 calculateCorrelation()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateCorrelation (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateCorrelation

Parameters

matrix	GLC-matrix

The correlation shows the correlation between different grey values

10.2.2.9 calculateDiffAverage()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateDiffAverage ( )
```

calculateDiffAverage

The difference average is defined as the weighted average of the diagonal probabilities.
In this way the relationship between occurences of pairs with similar intensity values and differing intensity values is measured.

10.2.2.10 calculateDiffEntropy()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateDiffEntropy ( )
```

calculateDiffEntropy

The difference entropy is defined as the entropy of the diagonal probabilities.
The different entropy feature gives information about the variablity in the differences of neighborhood intensities.

10.2.2.11 calculateDiffVariance()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateDiffVariance (
            T diffAvg )
```

calculateDiffVariance

Parameters

diffAvg	difference average The difference variance is defined as the weighted variance of the diagonal probabilities. It is a measure of heterogeneity. The more the intensity values are deviating from the mean, the higher is weight set to the intensity.

10.2.2.12 calculateDissimilarity()

```
template<class T , size_t R>
```

```
void GLCMFeatures< T, R >::calculateDissimilarity (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateDissimilarity

Parameters

matrix	GLC-matrix

The calculation of the dissimilarity is similar to the calculation of the contrast
It represents the mean difference of the intensity values between neighboring pixels/voxels

10.2.2.13 calculateFirstMCorrelation()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateFirstMCorrelation (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateFirstMCorrelation

Parameters

matrix	GLC-matrix

The first moment of correlation measures as well the homogeneity

10.2.2.14 calculateInverseDiff()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateInverseDiff (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateInverseDiff

Parameters

matrix	GLC-matrix

The calculation of the inverse difference is the weighted sum of the GLC-matrix elements
The bigger the difference between the intensities of the neighboring pixels/voxels, the smaller the weight
The more equal the neighbor pairs are, the lower is the denominator and the higher is the value

10.2.2.15 calculateInverseDiffMom()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateInverseDiffMom (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateInverseDiffMom

Parameters

matrix	GLC-matrix

The calculation of the inverse difference moment is similar to the inverse difference
It is also a weighted sum of the elements of the GLC-matrix. The higher the intensity values of a neighbor pair differ, the smaller is the weight.

10.2.2.16 calculateInverseDiffMomNorm()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateInverseDiffMomNorm (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateInverseDiffMomNorm

Parameters

matrix	GLC-matrix

The calculation of the inverse difference moment norm is similar to the one of inverse difference moment
Here the difference of the intensities of neighbor pairs is normalised by the number of grey levels

10.2.2.17 calculateInverseDiffNorm()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateInverseDiffNorm (
            boost::multi_array< double, 2 > glcMatrix,
            T inverseDiff )
```

calculateInverseDiffNorm

Parameters

matrix	GLC-matrix

The calculation of the inverse difference norm is similar to the inverse difference
The difference of the intensity differences is here normalised by the number of different grey levels

10.2.2.18 calculateInverseVariance()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateInverseVariance (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateInverseVariance

Parameters

matrix	GLC-matrix

The inverse variance is another measure if the image is locally homogen or not.

10.2.2.19 calculateJointEntropy()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateJointEntropy (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateJointEntropy

Parameters

matrix	GLC-matrix

Calculates the joint entropy
The joint entropy is a measurement for the uncertainity of numbers of neighbor pairs

10.2.2.20 calculateJointMaximum()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateJointMaximum (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateJointMaximum

Parameters

matrix	GLC-matrix

The joint maximum is the probability belonging to the neighbor pair which occurs the most in the VOI

10.2.2.21 calculateJointVariance()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateJointVariance (
            boost::multi_array< double, 2 > glcMatrix,
            T jointAvg )
```

calculateJointVariance

Parameters

matrix	GLC-matrix
jointAvg	joint average

Calculates the joint variance using the joint average defined before
The joint variance is the variance of the numbers of neighbor pairs

10.2.2.22 calculateMeanColProb()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateMeanColProb (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateMeanColProb

Parameters

matrix	glcMatrix: GLC-matrix

Calculate the mean column probability

10.2.2.23 calculateMeanRowProb()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateMeanRowProb (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateMeanRowProb

Parameters

matrix	glcMatrix: GLC-matrix

Calculate the mean row probability

10.2.2.24 calculateRowProb()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateRowProb (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateRowProb

Parameters

matrix	glcMatrix: GLC-matrix

Calculate the sum of the row probabilities and store them in vector sumProbRows

formula see above

10.2.2.25 calculateSecondMCorrelation()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateSecondMCorrelation (
            boost::multi_array< double, 2 > glcMatrix )
```

calculateSecondMCorrelation

Parameters

matrix	GLC-matrix

The second moment of correlation measures the similarity in intensity values for neighbor pairs

10.2.2.26 calculateSumAverage()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateSumAverage ( )
```

calculateSumAverage

This method measures the relationship between occurences of low intensity pairs and high intensity pairs.

10.2.2.27 calculateSumVariance()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::calculateSumVariance (
            T sumAvg )
```

calculateSumVariance

Parameters

sumAvg	sum The sum variance is defined as the weighted variance of the cross probabilities. Also this is a measurement of heterogeneity. The more the neighboring level pairs deviate from the mean, the higher they are weighted.

10.2.2.28 getCrossProbabilities()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::getCrossProbabilities (
            boost::multi_array< double, 2 > & glcMatrix )
```

getCrossProbabilities

Parameters

matrix	GLC-matrix

Calculate the cross probabilities and store them in vector crossProbabilities

10.2.2.29 getDiagonalProbabilities()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::getDiagonalProbabilities (
            boost::multi_array< double, 2 > & glcMatrix )
```

getDiagonalProbabilities

Parameters

matrix	glcMatrix: GLC-matrix

Calculate the diagonal probabilities and store them in vector diagonalProbabilities

formula see above

10.2.2.30 getNeighbours2D()

```
template<class T , size_t R>
std::vector< std::pair< T, T > > GLCMFeatures< T, R >::getNeighbours2D (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            int directionX,
            int directionY )
```

The method getNeighbors2D stores all neighbor pairs for the desired angle and the actual input matrix in a vector

Parameters

in	*inputMatrix*	the original matrix of the VOI
in	*angle*	: the actual angle
out	*neighbors*	vector containing all neighbor pairs of the actual direction

The method works as follows:
It looks for every matrix elements at the neighbors of the desired direction and makes a pair of the actual matrix element and its neighbors. These pairs are stored in a vector and are returned.

10.2.2.31 getNeighbours3D()

```
template<class T , size_t R>
std::vector< std::pair< T, T > > GLCMFeatures< T, R >::getNeighbours3D (
            boost::multi_array< T, R > inputMatrix,
            int angle,
            int directionZ )
```

The method getNeighbors3D stores all neighbor pairs for the desired angle and the actual input matrix in a vector

Parameters

in	*inputMatrix*	the original matrix of the VOI
in	*angle*	: the actual angle
in	*directionZ*	goes in the z-direction, adds the 3D calculation
out	*neighbors*	vector containing all neighbor pairs of the actual direction The method works as follows: It looks for every matrix elements at the neighbors of the desired direction and makes a pair of the actual matrix element and its neighbors. These pairs are stored in a vector and are returned.

10.2.2.32 getXYDirections()

```
template<class T , size_t R>
void GLCMFeatures< T, R >::getXYDirections (
            int & directionX,
            int & directionY,
            int angle )
```

getXYDirections

Parameters

int	directionX
int	directionY
int	angle

The function gets directionX and directionY as reference. Depending on the angle value, the parameter are set:
angle == 180 : go one pixel/voxel in x-direction; no move in y-direction
angle == 90 : no move in x-direction; go one pixel/voxel in y-direction
angle == 45 : go one pixel/voxel in x-direction; go one pixel/voxel in y direction
angle == 135 : go minus one pixel/voxel in x-direction; one pixel/voxel in y direction

The documentation for this class was generated from the following file:

- GLCMFeatures.h

10.3 GLCMFeatures2DAVG< T, R > Class Template Reference

`#include <GLCMFeatures2DAVG.h>`

Inheritance diagram for GLCMFeatures2DAVG< T, R >:

Collaboration diagram for GLCMFeatures2DAVG< T, R >:

Public Member Functions

- void **calculateAllGLCMFeatures2DAVG** (GLCMFeatures2DAVG< T, R > &glcmFeat, boost::multi_array< T, R > inputMatrix, float maxIntensity)
- void **writeCSVFileGLCM2DAVG** (GLCMFeatures2DAVG< T, R > glcmFeat, string outputFolder)
- void **writeOneFileGLCM2DAVG** (GLCMFeatures2DAVG< T, R > glcmFeat, string outputFolder)

Private Member Functions

- void **extractGLCMDataAVG** (vector< T > &glcmData, GLCMFeatures2DAVG< T, R > glcmFeatures)
- void fill2DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glcMatrix, int depth, int angle)
- boost::multi_array< double, 2 > calculateMatrix2DAVG (boost::multi_array< T, R > inputMatrix, int depth, int angle)

Private Attributes

- GLCMFeatures< T, R > **glcmComb**
- int **sizeMatrix**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.3.1 Detailed Description

template<class T, size_t R = 3>
class GLCMFeatures2DAVG< T, R >

The class GLCMFeatures2DAVG inherits from the matrix GLCMFeatures.
It does not merge the matrices before feature calculation.
For every slice a GLCMatrix is calculated and from every of this matrices all features are extracted.
Then the average value of all features is calculated.

10.3.2 Member Function Documentation

10.3.2.1 calculateMatrix2DAVG()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLCMFeatures2DAVG< T, R >::calculateMatrix2DAVG (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            int angle )  [private]
```

In the method calculateMatrix the GLCM-matrices for every direction are calculated, summed up and in the end the sum of this matrices is divided by the sum of the elements (= nr. of neighbor pairs) to obtain a matrix which contains the probabilities for the occurence of every neighbor pair.

Parameters

in		

10.3.2.2 fill2DMatrices()

```
template<class T , size_t R>
void GLCMFeatures2DAVG< T, R >::fill2DMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glcMatrix,
            int depth,
            int angle )  [private]
```

In the method fill2DMatrices the matrix is filled for all directions

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLCMFeatures2DAVG.h

10.4 GLCMFeatures2DFullMerge< T, R > Class Template Reference

`#include <GLCMFeatures2DFullMerge.h>`

Inheritance diagram for GLCMFeatures2DFullMerge< T, R >:

Collaboration diagram for GLCMFeatures2DFullMerge< T, R >:

Public Member Functions

- void **calculateAllGLCMFeatures2DFullMerge** (GLCMFeatures2DFullMerge< T, R > &glcmFeat, boost::multi_array< T, R > inputMatrix, float maxIntensity, vector< double > spacing, ConfigFile config)
- void **writeCSVFileGLCM2DFullMerge** (GLCMFeatures2DFullMerge< T, R > glcmFeat, string outputFolder)
- void **writeOneFileGLCM2DFullMerge** (GLCMFeatures2DFullMerge< T, R > glcmFeat, string outputFolder)

Private Member Functions

- void **extractGLCMDataFullMerge** (vector< T > &glcmData, GLCMFeatures2DFullMerge< T, R > glcmFeatures)
- void fill2DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glcMatrix, int depth, int angle)
- std::vector< std::pair< T, T > > getNeighbours2D (boost::multi_array< T, R > inputMatrix, int depth, int directionX, int directionY)
- boost::multi_array< double, 2 > calculateMatrix2DFullMerge (boost::multi_array< T, R > inputMatrix, float maxIntensity)

Private Attributes

- GLCMFeatures< T, R > **glcmComb**
- string **normGLCM**
- vector< double > **actualSpacing**
- int **N_g**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.4.1 Detailed Description

template<class T, size_t R = 3>
class GLCMFeatures2DFullMerge< T, R >

The class GLCMFeatures2DFullMerge herites from the class GLCMFeatures.
It merges the matrices of all slices and calculates afterwards the features from the merged matrix.
All feature calculations are defined in the class GLCMFeatures.
This class only contains the calculations of the merged matrix.

10.4.2 Member Function Documentation

10.4.2.1 calculateMatrix2DFullMerge()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLCMFeatures2DFullMerge< T, R >::calculateMatrix2DFullMerge (
            boost::multi_array< T, R > inputMatrix,
            float maxIntensity )  [private]
```

In the method calculateMatrix the GLCM-matrices for every direction are calculated, summed up and in the end the sum of this matrices is divided by the sum of the elements (= nr. of neighbor pairs) to obtain a matrix which contains the probabilities for the occurence of every neighbor pair.

Parameters

in		

10.4.2.2 fill2DMatrices()

```
template<class T , size_t R>
```

```
void GLCMFeatures2DFullMerge< T, R >::fill2DMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glcMatrix,
            int depth,
            int angle )  [private]
```

In the method fill2DMatrices the matrix is filled for all directions

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.4.2.3 getNeighbours2D()

```
template<class T , size_t R>
std::vector< std::pair< T, T > > GLCMFeatures2DFullMerge< T, R >::getNeighbours2D (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            int directionX,
            int directionY )  [private]
```

The method getNeighbors2D stores all neighbor pairs for the desired angle and the actual input matrix in a vector

Parameters

in	*inputMatrix*	the original matrix of the VOI
in	*angle*	: the actual angle
out	*neighbors*	vector containing all neighbor pairs of the actual direction

The method works as follows:
It looks for every matrix elements at the neighbors of the desired direction and makes a pair of the actual matrix element and its neighbors. These pairs are stored in a vector and are returned.

The documentation for this class was generated from the following file:

- GLCMFeatures2DFullMerge.h

10.5 GLCMFeatures2DMRG< T, R > Class Template Reference

```
#include <GLCMFeatures2DMRG.h>
```

Inheritance diagram for GLCMFeatures2DMRG< T, R >:

Collaboration diagram for GLCMFeatures2DMRG< T, R >:

Public Member Functions

- void **calculateAllGLCMFeatures2DMRG** (GLCMFeatures2DMRG< T, R > &glcmFeat, boost::multi_array< T, R > inputMatrix, float maxIntensity, vector< double > spacing, ConfigFile config)
- void **writeCSVFileGLCM2DMRG** (GLCMFeatures2DMRG< T, R > glcmFeat, string outputFolder)
- void **writeOneFileGLCM2DMRG** (GLCMFeatures2DMRG< T, R > glcmFeat, string outputFolder)

Private Member Functions

- void **extractGLCMDataMRG** (vector< T > &glcmData, GLCMFeatures2DMRG< T, R > glcmFeatures)
- void fill2DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glcMatrix, int depth, int angle)
- boost::multi_array< double, 2 > calculateMatrix2DMRG (boost::multi_array< T, R > inputMatrix, int depth, float maxIntensity)

Private Attributes

- string **normGLCM**
- vector< double > **actualSpacing**
- GLCMFeatures< T, R > **glcmComb**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.5.1 Detailed Description

template<class T, size_t R = 3>
class GLCMFeatures2DMRG< T, R >

The class GLCMFeatures2DMRG herites from the class GLCMFeatures.
It merges the matrices of every slice separately and calculates afterwards the features from the merged matrix.
Afterwards the mean value of the features for every slice is calculated.
The difference between this class and the other GLCMFeature-classes is only the type of merging of the matrix.
All feature calculations are defined in the class GLCMFeatures.
This class only contains the calculations of the merged matrix.

10.5.2 Member Function Documentation

10.5.2.1 calculateMatrix2DMRG()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLCMFeatures2DMRG< T, R >::calculateMatrix2DMRG (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            float maxIntensity )  [private]
```

In the method calculateMatrix the GLCM-matrices for every direction are calculated, summed up and in the end the sum of this matrices is divided by the sum of the elements (= nr. of neighbor pairs) to obtain a matrix which contains the probabilities for the occurence of every neighbor pair.

Parameters

in		

10.5.2.2 fill2DMatrices()

```
template<class T , size_t R>
void GLCMFeatures2DMRG< T, R >::fill2DMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glcMatrix,
            int depth,
            int angle )  [private]
```

In the method fill2DMatrices the matrix is filled for all directions

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLCMFeatures2DMRG.h

10.6 GLCMFeatures2DVMRG< T, R > Class Template Reference

`#include <GLCMFeatures2DVMRG.h>`

Inheritance diagram for GLCMFeatures2DVMRG< T, R >:

Collaboration diagram for GLCMFeatures2DVMRG< T, R >:

Public Member Functions

- void **calculateAllGLCMFeatures2DVMRG** (GLCMFeatures2DVMRG< T, R > &glcmFeat, boost::multi_↩ array< T, R > inputMatrix, float maxIntensity, vector< double > spacing, ConfigFile config)
- void **writeCSVFileGLCM2DVMRG** (GLCMFeatures2DVMRG< T, R > glcmFeat, string outputFolder)
- void **writeOneFileGLCM2DVMRG** (GLCMFeatures2DVMRG< T, R > glcmFeat, string outputFolder)

Private Member Functions

- void **extractGLCMDataVMRG** (vector< T > &glcmData, GLCMFeatures2DVMRG< T, R > glcmFeatures)
- void fill2DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glcMatrix, int depth, int angle)
- boost::multi_array< double, 2 > calculateMatrix2DVMRG (boost::multi_array< T, R > inputMatrix, float maxIntensity)

Private Attributes

- GLCMFeatures< T, R > **glcmComb**
- string **normGLCM**
- vector< double > **actualSpacing**
- int **N_g**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.6.1 Detailed Description

template<class T, size_t R = 3>
class GLCMFeatures2DVMRG< T, R >

The class GLCMFeatures2DVMRG herites from the class GLCMFeatures.
It merges the matrices of all slices and calculates afterwards the features from the merged matrix.
All feature calculations are defined in the class GLCMFeatures.
This class only contains the calculations of the merged matrix.

10.6.2 Member Function Documentation

10.6.2.1 calculateMatrix2DVMRG()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLCMFeatures2DVMRG< T, R >::calculateMatrix2DVMRG (
            boost::multi_array< T, R > inputMatrix,
            float maxIntensity )  [private]
```

In the method calculateMatrix the GLCM-matrices for every direction are calculated, summed up and in the end the sum of this matrices is divided by the sum of the elements (= nr. of neighbor pairs) to obtain a matrix which contains the probabilities for the occurence of every neighbor pair.

Parameters

in		

10.6.2.2 fill2DMatrices()

```
template<class T , size_t R>
```

```
void GLCMFeatures2DVMRG< T, R >::fill2DMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glcMatrix,
            int depth,
            int angle )  [private]
```

In the method fill2DMatrices the matrix is filled for all directions

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLCMFeatures2DVMRG.h

10.7 GLCMFeatures2DWMerge< T, R > Class Template Reference

`#include <GLCMFeatures2DWMerge.h>`

Inheritance diagram for GLCMFeatures2DWMerge< T, R >:

Collaboration diagram for GLCMFeatures2DWMerge< T, R >:

Public Member Functions

- void **calculateAllGLCMFeatures2DWMerge** (GLCMFeatures2DWMerge< T, R > &glcmFeat, boost::multi_array< T, R > inputMatrix, float maxIntensity, vector< double > spacing, ConfigFile config)
- void **writeCSVFileGLCM2DWMerge** (GLCMFeatures2DWMerge< T, R > glcmFeat, string outputFolder)
- void **writeOneFileGLCM2DWMerge** (GLCMFeatures2DWMerge< T, R > glcmFeat, string outputFolder)

Private Member Functions

- void **extractGLCMDataWMerge** (vector< T > &glcmData, GLCMFeatures2DWMerge< T, R > glcmFeatures)
- void fill2DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glcMatrix, int depth, int angle)
- std::vector< std::pair< T, T > > getNeighbours2D (boost::multi_array< T, R > inputMatrix, int depth, int directionX, int directionY)
- boost::multi_array< double, 2 > calculateMatrix2DWMerge (boost::multi_array< T, R > inputMatrix, int depth, float maxIntensity)

Private Attributes

- string **normGLCM**
- vector< double > **actualSpacing**
- GLCMFeatures< T, R > **glcmComb**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.7.1 Detailed Description

template<class T, size_t R = 3>
class GLCMFeatures2DWMerge< T, R >

The class GLCMFeatures2DWMerge herites from the class GLCMFeatures.
It merges the matrices of every slice separately and calculates afterwards the features from the merged matrix.
Afterwards the mean value of the features for every slice is calculated.
The difference between this class and the other GLCMFeature-classes is only the type of merging of the matrix.
All feature calculations are defined in the class GLCMFeatures.
This class only contains the calculations of the merged matrix.

10.7.2 Member Function Documentation

10.7.2.1 calculateMatrix2DWMerge()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLCMFeatures2DWMerge< T, R >::calculateMatrix2DWMerge (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            float maxIntensity )  [private]
```

In the method calculateMatrix the GLCM-matrices for every direction are calculated, summed up and in the end the sum of this matrices is divided by the sum of the elements (= nr. of neighbor pairs) to obtain a matrix which contains the probabilities for the occurence of every neighbor pair.

Parameters

in		

10.7.2.2 fill2DMatrices()

```
template<class T , size_t R>
void GLCMFeatures2DWMerge< T, R >::fill2DMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glcMatrix,
            int depth,
            int angle )  [private]
```

In the method fill2DMatrices the matrix is filled for all directions

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.7.2.3 getNeighbours2D()

```
template<class T , size_t R>
std::vector< std::pair< T, T > > GLCMFeatures2DWMerge< T, R >::getNeighbours2D (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            int directionX,
            int directionY )  [private]
```

The method getNeighbors2D stores all neighbor pairs for the desired angle and the actual input matrix in a vector

Parameters

in	*inputMatrix*	the original matrix of the VOI
in	*angle*	: the actual angle
out	*neighbors*	vector containing all neighbor pairs of the actual direction

The method works as follows:\ It looks for every matrix elements at the neighbors of the desired direction and makes a pair of the actual matrix element and its neighbors. These pairs are stored in a vector and are returned.

The documentation for this class was generated from the following file:

- GLCMFeatures2DWMerge.h

10.8 GLCMFeatures2DWOMerge< T, R > Class Template Reference

`#include <GLCMFeatures2DWOMerge.h>`

Inheritance diagram for GLCMFeatures2DWOMerge< T, R >:

Collaboration diagram for GLCMFeatures2DWOMerge< T, R >:

Public Member Functions

- void **calculateAllGLCMFeatures2DWOMerge** (GLCMFeatures2DWOMerge< T, R > &glcmFeat, boost::multi_array< T, R > inputMatrix, float maxIntensity)
- void **writeCSVFileGLCM2DWOMerge** (GLCMFeatures2DWOMerge< T, R > glcmFeat, string outputFolder)
- void **writeOneFileGLCM2DWOMerge** (GLCMFeatures2DWOMerge< T, R > glcmFeat, string outputFolder)

Private Member Functions

- void **extractGLCMDataWOMerge** (vector< T > &glcmData, GLCMFeatures2DWOMerge< T, R > glcmFeatures)
- void fill2DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glcMatrix, int depth, int angle)
- std::vector< std::pair< T, T > > getNeighbours2D (boost::multi_array< T, R > inputMatrix, int depth, int angle)
- boost::multi_array< double, 2 > calculateMatrix2DWOMerge (boost::multi_array< T, R > inputMatrix, int depth, int angle)

Private Attributes

- GLCMFeatures< T, R > **glcmComb**
- int **sizeMatrix**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.8.1 Detailed Description

template<class T, size_t R = 3>
class GLCMFeatures2DWOMerge< T, R >

The class GLCMFeatures2DWOMerge inherits from the matrix GLCMFeatures.
It does not merge the matrices before feature calculation.
For every slice a GLCMatrix is calculated and from every of this matrices all features are extracted.
Then the average value of all features is calculated.

10.8.2 Member Function Documentation

10.8.2.1 calculateMatrix2DWOMerge()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLCMFeatures2DWOMerge< T, R >::calculateMatrix2DWOMerge (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            int angle )  [private]
```

In the method calculateMatrix the GLCM-matrices for every direction are calculated, summed up and in the end the sum of this matrices is divided by the sum of the elements (= nr. of neighbor pairs) to obtain a matrix which contains the probabilities for the occurence of every neighbor pair.

Parameters

in		

10.8.2.2 fill2DMatrices()

```
template<class T , size_t R>
void GLCMFeatures2DWOMerge< T, R >::fill2DMatrices (
```

```
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glcMatrix,
            int depth,
            int angle )  [private]
```

In the method fill2DMatrices the matrix is filled for all directions

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.8.2.3 getNeighbours2D()

```
template<class T , size_t R>
std::vector< std::pair< T, T > > GLCMFeatures2DWOMerge< T, R >::getNeighbours2D (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            int angle )  [private]
```

The method getNeighbors2D stores all neighbor pairs for the desired angle and the actual input matrix in a vector

Parameters

in	*inputMatrix*	the original matrix of the VOI
in	*angle*	: the actual angle
out	*neighbors*	vector containing all neighbor pairs of the actual direction

The method works as follows:\ It looks for every matrix elements at the neighbors of the desired direction and makes a pair of the actual matrix element and its neighbors. These pairs are stored in a vector and are returned.

The documentation for this class was generated from the following file:

- GLCMFeatures2DWOMerge.h

10.9 GLCMFeatures3D< T, R > Class Template Reference

`#include <GLCMFeatures3D.h>`

Inheritance diagram for GLCMFeatures3D< T, R >:

Collaboration diagram for GLCMFeatures3D< T, R >:

Public Member Functions

- void **writeCSVFileGLCM3D** (GLCMFeatures3D< T, R > glcmFeat)
- void **calculateAllGLCMFeatures3D** (GLCMFeatures3D< T, R > &glcmFeat, boost::multi_array< T, R > inputMatrix)

Private Types

- typedef boost::multi_array< double, 2 > **glcmat**

Private Member Functions

- void **createGLCMMatrix** (boost::multi_array< T, R > inputMatrix)
- void **defineGLCMFeatures3D** (vector< string > &features)
- void **extractGLCMData3D** (vector< T > &glcmData, GLCMFeatures3D< T, R > glcmFeatures)
- void **generate3DMatrices** (boost::multi_array< T, R > inputMatrix, glcmat &GLCM3D, int angle, int directionZ)
- std::vector< std::pair< T, T > > **getNeighbours3D** (boost::multi_array< T, R > inputMatrix, int angle, int directionZ)
- void **fill3DMatrices** (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glcMatrix, int angle, int directionZ)
- boost::multi_array< double, 2 > **getMatrixSum** (boost::multi_array< T, R > inputMatrix)

Private Attributes

- vector< T > **diffGreyLevels**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- Image< T, R > **image** {Image<T,R>(4,5,4)}
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.9.1 Detailed Description

template<class T, size_t R>
class GLCMFeatures3D< T, R >

In the class GLCMFeatures3D the calculation of the GLC-matrix is modified to 3D mode.
So now every voxel has 13 direct neighbors.
All feature calculations stay the same as in the 2D approach.

The documentation for this class was generated from the following file:

- GLCMFeatures3D.h

10.10 GLCMFeatures3DAVG< T, R > Class Template Reference

`#include <GLCMFeatures3DAVG.h>`

Inheritance diagram for GLCMFeatures3DAVG< T, R >:

Collaboration diagram for GLCMFeatures3DAVG< T, R >:

Public Member Functions

- void **writeCSVFileGLCM3DAVG** (GLCMFeatures3DAVG< T, R > glcmFeat, string outputFolder)
- void **writeOneFileGLCM3DAVG** (GLCMFeatures3DAVG< T, R > glcmFeat, string outputFolder)
- void **calculateAllGLCMFeatures3DAVG** (GLCMFeatures3DAVG< T, R > &glcmFeat, boost::multi_array< T, R > inputMatrix, float maxIntensity)

Private Types

- typedef boost::multi_array< double, 2 > **glcmat**

Private Member Functions

- void **defineGLCMFeatures3DAVG** (vector< string > &features)
- void **extractGLCMData3D** (vector< T > &glcmData, GLCMFeatures3DAVG< T, R > glcmFeatures)
- void fill3DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &sum, int angle, int directionZ)
- boost::multi_array< double, 2 > getMatrixSum (boost::multi_array< T, R > inputMatrix, int ang, int directionZ)

Private Attributes

- GLCMFeatures< T, R > **glcm**
- int **sizeMatrix**
- vector< T > **diffGreyLevels**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.10.1 Detailed Description

template<class T, size_t R>
class GLCMFeatures3DAVG< T, R >

The class GLCMFeatures3DAVG herites from the class GLCMFeatures.
It considers 13 neighbors to calculate the cooccurrence features.
It calculates a GLCM matrix for every angle and extracts the feature from every matrix. Then the mean value over all these features is calculated.
All feature calculations are defined in the class GLCMFeatures.
This class only contains the calculations of the merged matrix.

10.10.2 Member Function Documentation

10.10.2.1 fill3DMatrices()

```
template<class T , size_t R>
void GLCMFeatures3DAVG< T, R >::fill3DMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glcMatrix,
            int angle,
            int directionZ )  [private]
```

In the method fill3DMatrices the matrix is filled for all directions

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.10.2.2 getMatrixSum()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLCMFeatures3DAVG< T, R >::getMatrixSum (
            boost::multi_array< T, R > inputMatrix,
            int ang,
            int directionZ )  [private]
```

In the method getMatrixSum calculates the sum from one GLCM matrix and its inverse

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLCMFeatures3DAVG.h

10.11 GLCMFeatures3DMRG< T, R > Class Template Reference

`#include <GLCMFeatures3DMRG.h>`

Inheritance diagram for GLCMFeatures3DMRG< T, R >:

Collaboration diagram for GLCMFeatures3DMRG< T, R >:

Public Member Functions

- void **writeCSVFileGLCM3DMRG** (GLCMFeatures3DMRG< T, R > glcmFeat, string outputFolder)
- void **writeOneFileGLCM3DMRG** (GLCMFeatures3DMRG< T, R > glcmFeat, string outputFolder)
- void **calculateAllGLCMFeatures3DMRG** (GLCMFeatures3DMRG< T, R > &glcmFeat, boost::multi_array< T, R > inputMatrix, float maxIntensity, vector< double > spacing, ConfigFile config)

Private Types

- typedef boost::multi_array< double, 2 > **glcmat**

Private Member Functions

- void **defineGLCMFeatures3DMRG** (vector< string > &features)
- void **extractGLCMData3D** (vector< T > &glcmData, GLCMFeatures3DMRG< T, R > glcmFeatures)
- void fill3DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glcMatrix, int angle, int directionZ)
- boost::multi_array< double, 2 > getMatrixSum (boost::multi_array< T, R > inputMatrix, float maxIntensity)

Private Attributes

- string **normGLCM**
- vector< double > **actualSpacing**
- vector< T > **diffGreyLevels**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.11.1 Detailed Description

template< class T, size_t R >
class GLCMFeatures3DMRG< T, R >

The class GLCMFeatures3DMRG herites from the class GLCMFeatures.
It considers 13 neighbors to calculate the cooccurrence features.
It calculates the feature values for every angle and calculates then the mean value of the features.
All feature calculations are defined in the class GLCMFeatures.
This class only contains the calculations of the merged matrix.

10.11.2 Member Function Documentation

10.11.2.1 fill3DMatrices()

```
template<class T , size_t R>
void GLCMFeatures3DMRG< T, R >::fill3DMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glcMatrix,
            int angle,
            int directionZ )  [private]
```

In the method fill3DMatrices the GLCM matrix is filled with the according values

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.11.2.2 getMatrixSum()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLCMFeatures3DMRG< T, R >::getMatrixSum (
            boost::multi_array< T, R > inputMatrix,
            float maxIntensity ) [private]
```

In the method getMatrixSum calculates the sum of all calculated GLCM matrices

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLCMFeatures3DMRG.h

10.12 GLCMFeatures3DWMerge< T, R > Class Template Reference

```
#include <GLCMFeatures3DWMerge.h>
```

Inheritance diagram for GLCMFeatures3DWMerge< T, R >:

Collaboration diagram for GLCMFeatures3DWMerge< T, R >:

Public Member Functions

- std::vector< std::pair< T, T > > getNeighbours3D (boost::multi_array< T, R > inputMatrix, int angle, int directionZ)
- void **writeCSVFileGLCM3DWMerge** (GLCMFeatures3DWMerge< T, R > glcmFeat, string outputFolder)
- void **writeOneFileGLCM3DWMerge** (GLCMFeatures3DWMerge< T, R > glcmFeat, string outputFolder)
- void **calculateAllGLCMFeatures3DWMerge** (GLCMFeatures3DWMerge< T, R > &glcmFeat, boost::multi_array< T, R > inputMatrix, float maxIntensity)

Private Types

- typedef boost::multi_array< double, 2 > **glcmat**

Private Member Functions

- void **defineGLCMFeatures3DWMerge** (vector< string > &features)
- void **extractGLCMData3D** (vector< T > &glcmData, GLCMFeatures3DWMerge< T, R > glcmFeatures)
- void fill3DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &sum, int angle, int directionZ)
- boost::multi_array< double, 2 > getMatrixSum (boost::multi_array< T, R > inputMatrix, int ang, int directionZ)

Private Attributes

- int **sizeMatrix**
- vector< T > **diffGreyLevels**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.12.1 Detailed Description

template<class T, size_t R>
class GLCMFeatures3DWMerge< T, R >

The class GLCMFeatures3DWMerge herites from the class GLCMFeatures.
It considers 13 neighbors to calculate the cooccurrence features.
It calculates the feature values for every angle and calculates then the mean value of the features.
All feature calculations are defined in the class GLCMFeatures.
This class only contains the calculations of the merged matrix.

10.12.2 Member Function Documentation

10.12.2.1 fill3DMatrices()

```
template<class T , size_t R>
void GLCMFeatures3DWMerge< T, R >::fill3DMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glcMatrix,
            int angle,
            int directionZ )  [private]
```

In the method fill3DMatrices the matrix is filled for all directions

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.12.2.2 getMatrixSum()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLCMFeatures3DWMerge< T, R >::getMatrixSum (
            boost::multi_array< T, R > inputMatrix,
            int ang,
            int directionZ )  [private]
```

In the method getMatrixSum calculates the sum from one GLCM matrix and its inverse

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.12.2.3 getNeighbours3D()

```
template<class T , size_t R>
std::vector< std::pair< T, T > > GLCMFeatures3DWMerge< T, R >::getNeighbours3D (
            boost::multi_array< T, R > inputMatrix,
            int angle,
            int directionZ )
```

The method getNeighbors3D stores all neighbor pairs for the desired angle and the actual input matrix in a vector

Parameters

in	*inputMatrix*	the original matrix of the VOI
in	*angle*	: the actual angle
in	*directionZ*	goes in the z-direction, adds the 3D calculation
out	*neighbors*	vector containing all neighbor pairs of the actual direction The method works as follows: It looks for every matrix elements at the neighbors of the desired direction and makes a pair of the actual matrix element and its neighbors. These pairs are stored in a vector and are returned.

The documentation for this class was generated from the following file:

- GLCMFeatures3DWMerge.h

10.13 GLCMFeatures3DWOMerge< T, R > Class Template Reference

`#include <GLCMFeatures3DWOMerge.h>`

Inheritance diagram for GLCMFeatures3DWOMerge< T, R >:

Collaboration diagram for GLCMFeatures3DWOMerge< T, R >:

Public Member Functions

- void **writeCSVFileGLCM3D** (GLCMFeatures3DWOMerge< T, R > glcmFeat, string outputFolder)
- void **writeOneFileGLCM3D** (GLCMFeatures3DWOMerge< T, R > glcmFeat, string outputFolder)
- void **calculateAllGLCMFeatures3DWOMerge** (GLCMFeatures3DWOMerge< T, R > &glcmFeat, boost↩ ::multi_array< T, R > inputMatrix, float maxIntensity, vector< double > spacing, ConfigFile config)

Private Types

- typedef boost::multi_array< double, 2 > **glcmat**

Private Member Functions

- void **defineGLCMFeatures3DWOMerge** (vector< string > &features)
- void **extractGLCMData3D** (vector< T > &glcmData, GLCMFeatures3DWOMerge< T, R > glcmFeatures)
- void fill3DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glcMatrix, int angle, int directionZ)
- boost::multi_array< double, 2 > getMatrixSum (boost::multi_array< T, R > inputMatrix, float maxIntensity)

Private Attributes

- string **normGLCM**
- vector< double > **actualSpacing**
- vector< T > **diffGreyLevels**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- T **HX**
- T **HXY**
- T **HXY1**
- T **HXY2**

10.13.1 Detailed Description

template<class T, size_t R>
class GLCMFeatures3DWOMerge< T, R >

The class GLCMFeatures3DWOMerge herites from the class GLCMFeatures.
It considers 13 neighbors to calculate the cooccurrence features.
It calculates a GLCM matrix for every angle and extracts the feature from every matrix. Then the mean value over all these features is calculated.
All feature calculations are defined in the class GLCMFeatures.
This class only contains the calculations of the merged matrix.

10.13.2 Member Function Documentation

10.13.2.1 fill3DMatrices()

```
template<class T , size_t R>
void GLCMFeatures3DWOMerge< T, R >::fill3DMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glcMatrix,
            int angle,
            int directionZ )  [private]
```

In the method fill3DMatrices the GLCM matrix is filled with the according values

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.13.2.2 getMatrixSum()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLCMFeatures3DWOMerge< T, R >::getMatrixSum (
            boost::multi_array< T, R > inputMatrix,
            float maxIntensity )  [private]
```

In the method getMatrixSum calculates the sum of all calculated GLCM matrices

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLCMFeatures3DWOMerge.h

10.14 GLDZMFeatures2D< T, R > Class Template Reference

```
#include <GLDZMFeatures2D.h>
```

Inheritance diagram for GLDZMFeatures2D< T, R >:

Collaboration diagram for GLDZMFeatures2D< T, R >:

Public Member Functions

- int **checkNeighbors** (boost::multi_array< T, R > distanceMap, boost::multi_array< T, R > inputMatrix, vector< int > actualIndex)
- void **defineGLDZMFeatures** (vector< string > &features)
- int getMinimalDistance (boost::multi_array< T, R > distanceMap, vector< vector< int > > matrixIndices)
- void **writeCSVFileGLDZM** (GLDZMFeatures2D< T, R > gldzmFeat, string outputFolder)
- void **writeOneFileGLDZM** (GLDZMFeatures2D< T, R > gldzmFeat, string outputFolder)
- void **calculateAllGLDZMFeatures2D** (GLDZMFeatures2D< T, R > &gldzmFeat, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< T > vectorMatElem, ConfigFile config)
- void **defineGLDZMFeatures** (vector< string > &features)
- int **getMinimalDistance** (boost::multi_array< T, R > distanceMap, vector< vector< int > > matrixIndices)
- void **writeCSVFileGLDZM** (GLDZMFeatures2D< T, R > gldzmFeat, string outputFolder)
- void **writeOneFileGLDZM** (GLDZMFeatures2D< T, R > gldzmFeat, string outputFolder)
- void **calculateAllGLDZMFeatures2D** (GLDZMFeatures2D< T, R > &gldzmFeat, boost::multi_array< T, R > distanceMap, Image< T, R > imageAttr, ConfigFile config)

Private Member Functions

- void **extractGLDZMData** (vector< T > &gldzmData, GLDZMFeatures2D< T, R > gldzmFeatures)
- boost::multi_array< T, R > generateDistanceMap (boost::multi_array< T, R > inputMatrix)
- boost::multi_array< double, 2 > fillMatrix (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &gldzmat)
- boost::multi_array< double, 2 > getMatrix (boost::multi_array< T, R > inputMatrix)
- void **extractGLDZMData** (vector< T > &gldzmData, GLDZMFeatures2D< T, R > gldzmFeatures)
- boost::multi_array< double, 2 > fillMatrix (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > distanceMap, boost::multi_array< double, 2 > &gldzmat)
- boost::multi_array< double, 2 > getMatrix (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > distanceMap)

Private Attributes

- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- int **totalNrZones**
- vector< double > **rowSums**
- vector< double > **colSums**
- GLSZMFeatures2D< T, R > **GLSZM2D**
- GLSZMFeatures2DMRG< T, R > **GLSZM2D**

Additional Inherited Members

10.14.1 Detailed Description

template<class T, size_t R = 3>
class GLDZMFeatures2D< T, R >

The class GLDZM is the class of the Grey Level Distance Zone Matrices.
It combines the grey level size zone matrices with a distance map. Voxels are considered as connected, when they have the same grey value.
The distance to the edge is also defined according to 4-connectedness. The distance of a voxel to the outer border is defined as the number of edges that have to be crossed to reach the edge of the VOI.

10.14.2 Member Function Documentation

10.14.2.1 fillMatrix() [1/2]

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLDZMFeatures2D< T, R >::fillMatrix (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > distanceMap,
            boost::multi_array< double, 2 > & gldzmat )  [private]
```

In the method fillMatrix the GLDZM matrix is filled, taking the original matrix of the VOI as input.
The GLDZM matrix is given as reference and filled in the function

Parameters

in		

10.14.2.2 fillMatrix() [2/2]

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLDZMFeatures2D< T, R >::fillMatrix (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & gldzmat )  [private]
```

In the method fillMatrix the GLDZM matrix is filled, taking the original matrix of the VOI as input.
The GLDZM matrix is given as reference and filled in the function

Parameters

in		

10.14.2.3 generateDistanceMap()

```
template<class T , size_t R>
boost::multi_array< T, R > GLDZMFeatures2D< T, R >::generateDistanceMap (
            boost::multi_array< T, R > inputMatrix )  [private]
```

In the method generateDistanceMap the distance map is generated, taking the matrix of the VOI as input.
According to the position of a value in the matrix, the corresponding distance is saved in the distance map

Parameters

in		

10.14.2.4 getMatrix() [1/2]

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLDZMFeatures2D< T, R >::getMatrix (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > distanceMap )  [private]
```

In the method getMatrix the GLDZM matrix is generated and filled using the function fillMatrix. The function is mainly used get the size of the GLDZM matrix.

Parameters

in		

10.14.2.5 getMatrix() [2/2]

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLDZMFeatures2D< T, R >::getMatrix (
            boost::multi_array< T, R > inputMatrix )  [private]
```

In the method getMatrix the GLDZM matrix is generated and filled using the function fillMatrix. The function is mainly used get the size of the GLDZM matrix.

Parameters

in		

10.14.2.6 getMinimalDistance()

```
template<class T , size_t R>
int GLDZMFeatures2D< T, R >::getMinimalDistance (
            boost::multi_array< T, R > distanceMap,
            vector< vector< int > > matrixIndices )
```

In the method getMinimalDistance gets the minimal distance of a zone to the VOI edge.

Parameters

in		

The documentation for this class was generated from the following files:

- GLDZMFeatures2D.h
- GLDZMFeatures2DMRG.h

10.15 GLDZMFeatures2DAVG< T, R > Class Template Reference

`#include <GLDZMFeatures2DAVG.h>`

Inheritance diagram for GLDZMFeatures2DAVG< T, R >:

Collaboration diagram for GLDZMFeatures2DAVG< T, R >:

Public Member Functions

- boost::multi_array< T, R > generateDistanceMap (boost::multi_array< T, R > inputMatrix, boost::multi_↩ array< T, R > &distanceMap)
- void **writeCSVFileGLDZM2DAVG** (GLDZMFeatures2DAVG< T, R > gldzmFeat, string outputFolder)
- void **writeOneFileGLDZM2DAVG** (GLDZMFeatures2DAVG< T, R > gldzmFeat, string outputFolder)
- void **calculateAllGLDZMFeatures2DAVG** (GLDZMFeatures2DAVG< T, R > &gldzmFeat, Image< T, R > imageAttr, boost::multi_array< T, R > distanceMap, ConfigFile config)

Private Member Functions

- int **checkNeighbors** (boost::multi_array< T, R > &distanceMap, boost::multi_array< T, R > &inputMatrix, vector< int > actIndex, int actualDistance)
- void **extractGLDZMData2DAVG** (vector< T > &gldzmData, GLDZMFeatures2DAVG< T, R > gldzm↩ Features)
- boost::multi_array< double, 2 > fillMatrix (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > distanceMap, boost::multi_array< double, 2 > &gldzmat, int depth)
- boost::multi_array< double, 2 > getMatrix (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > distanceMap, int depth)
- int **checkNeighborsNAN** (boost::multi_array< T, R > &inputMatrix, boost::multi_array< T, R > &temp↩ Matrix, vector< int > actIndex, int actDist)

Private Attributes

- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- int **totalNrZones**
- vector< double > **rowSums**
- vector< double > **colSums**
- GLSZMFeatures2DMRG< T, R > **GLSZM2D**
- GLDZMFeatures2D< T, R > **GLDZM2D**
- GLDZMFeatures3D< T, R > **GLDZM3D**

Additional Inherited Members

10.15.1 Detailed Description

template<class T, size_t R = 3>
class GLDZMFeatures2DAVG< T, R >

The class GLDZMFeatures2DAVG is the class of the Grey Level Distance Zone Matrices, it inheritates from the class GLDZMFeatures2D.
For further explanation look at GLDZMFeatures2D file.
For every slice a GLDZM is calculated and from every of this matrices all features are extracted.
Then the average value of all features is calculated.

10.15.2 Member Function Documentation

10.15.2.1 fillMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLDZMFeatures2DAVG< T, R >::fillMatrix (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > distanceMap,
            boost::multi_array< double, 2 > & gldzmat,
            int depth )  [private]
```

In the method fillMatrix the GLDZM matrix is filled, taking the original matrix of the VOI as input.
The GLDZM matrix is given as reference and filled in the function

Parameters

in		

10.15.2.2 generateDistanceMap()

```
template<class T , size_t R>
boost::multi_array< T, R > GLDZMFeatures2DAVG< T, R >::generateDistanceMap (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > & distanceMap )
```

In the method generateDistanceMap the distance map is generated, taking the matrix of the VOI as input.
According to the position of a value in the matrix, the corresponding distance is saved in the distance map

Parameters

in		

10.15.2.3 getMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLDZMFeatures2DAVG< T, R >::getMatrix (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > distanceMap,
            int depth )  [private]
```

In the method getMatrix the GLDZM matrix is generated and filled using the function fillMatrix. The function is mainly used get the size of the GLDZM matrix.

Parameters

in		

The documentation for this class was generated from the following file:

- GLDZMFeatures2DAVG.h

10.16 GLDZMFeatures2DWOMerge< T, R > Class Template Reference

```
#include <GLDZMFeatures2DWOMerge.h>
```

Inheritance diagram for GLDZMFeatures2DWOMerge< T, R >:

Collaboration diagram for GLDZMFeatures2DWOMerge< T, R >:

Public Member Functions

- void **writeCSVFileGLDZM2DWOMerge** (GLDZMFeatures2DWOMerge< T, R > gldzmFeat, string output↩ Folder)
- void **writeOneFileGLDZM2DWOMerge** (GLDZMFeatures2DWOMerge< T, R > gldzmFeat, string output↩ Folder)
- void **calculateAllGLDZMFeatures2DWOMerge** (GLDZMFeatures2DWOMerge< T, R > &gldzmFeat, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, ConfigFile config)

Private Member Functions

- void **extractGLDZMData2DWOMerge** (vector< T > &gldzmData, GLDZMFeatures2DWOMerge< T, R > gldzmFeatures)
- boost::multi_array< T, R > generateDistanceMap (boost::multi_array< T, R > inputMatrix)
- boost::multi_array< double, 2 > fillMatrix (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &gldzmat, int depth)
- boost::multi_array< double, 2 > getMatrix (boost::multi_array< T, R > inputMatrix, int depth)

Private Attributes

- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- int **totalNrZones**
- vector< double > **rowSums**
- vector< double > **colSums**
- GLSZMFeatures2D< T, R > **GLSZM2D**
- GLDZMFeatures2D< T, R > **GLDZM2D**

10.16.1 Detailed Description

template<class T, size_t R = 3>
class GLDZMFeatures2DWOMerge< T, R >

The class GLDZMFeatures2DWOMerge is the class of the Grey Level Distance Zone Matrices, it inheritates from the class GLDZMFeatures2D.
For further explanation look at GLDZMFeatures2D file.
For every slice a GLDZM is calculated and from every of this matrices all features are extracted.
Then the average value of all features is calculated.

10.16.2 Member Function Documentation

10.16.2.1 fillMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLDZMFeatures2DWOMerge< T, R >::fillMatrix (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & gldzmat,
            int depth )  [private]
```

In the method fillMatrix the GLDZM matrix is filled, taking the original matrix of the VOI as input.
The GLDZM matrix is given as reference and filled in the function

Parameters

in		

10.16.2.2 generateDistanceMap()

```
template<class T , size_t R>
boost::multi_array< T, R > GLDZMFeatures2DWOMerge< T, R >::generateDistanceMap (
            boost::multi_array< T, R > inputMatrix )  [private]
```

In the method generateDistanceMap the distance map is generated, taking the matrix of the VOI as input. According to the position of a value in the matrix, the corresponding distance is saved in the distance map

Parameters

in		

10.16.2.3 getMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLDZMFeatures2DWOMerge< T, R >::getMatrix (
            boost::multi_array< T, R > inputMatrix,
            int depth )  [private]
```

In the method getMatrix the GLDZM matrix is generated and filled using the function fillMatrix. The function is mainly used get the size of the GLDZM matrix.

Parameters

in		

The documentation for this class was generated from the following file:

- GLDZMFeatures2DWOMerge.h

10.17 GLDZMFeatures3D< T, R > Class Template Reference

```
#include <GLDZMFeatures3D.h>
```

Inheritance diagram for GLDZMFeatures3D< T, R >:

Collaboration diagram for GLDZMFeatures3D< T, R >:

Public Member Functions

- void **writeCSVFileGLDZM3D** (GLDZMFeatures3D< T, R > gldzmFeat, string outputFolder)
- void **writeOneFileGLDZM3D** (GLDZMFeatures3D< T, R > gldzmFeat, string outputFolder)
- void **calculateAllGLDZMFeatures3D** (GLDZMFeatures3D< T, R > &gldzmFeat, Image< T, R > imageAttr, ConfigFile config)
- int **checkNeighbors3DNAN** (boost::multi_array< T, R > distanceMap, boost::multi_array< T, R > &tempMatrix, vector< int > actIndex, int actDistance)
- int **checkNeighbors3D** (boost::multi_array< T, R > &distanceMap, boost::multi_array< T, R > &tempMatrix, vector< int > actIndex, int actualDistance)

Private Member Functions

- void **extractGLDZMData3D** (vector< T > &gldzmData, GLDZMFeatures3D< T, R > gldzmFeatures)
- boost::multi_array< T, R > **generateDistanceMap3D** (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > &distanceMap)
- void fillMatrix3D (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > distanceMatrix, boost::multi_array< double, 2 > &gldzmat)
- boost::multi_array< double, 2 > getMatrix3D (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > distanceMatrix)

Private Attributes

- GLSZMFeatures2DMRG< T, R > **GLSZM2D**
- GLDZMFeatures2D< T, R > **GLDZM2D**
- GLSZMFeatures3D< T, R > **GLSZM3D**
- vector< T > **diagonalProbabilities**
- vector< T > **crossProbabilities**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- int **totalNrZones**
- vector< double > **rowSums**
- vector< double > **colSums**

Additional Inherited Members

10.17.1 Detailed Description

template<class T, size_t R = 3>
class GLDZMFeatures3D< T, R >

The class GLDZM is the class of the Grey Level Distance Zone Matrices for the 3D approach.
It combines the grey level size zone matrices with a distance map. Voxels are considered as connected, when they have the same grey value.
The distance to the edge is also defined according to 6-connectedness. The distance of a voxel to the outer border is defined as the number of edges that have to be crossed to reach the edge of the VOI.

10.17.2 Member Function Documentation

10.17.2.1 fillMatrix3D()

```
template<class T , size_t R>
void GLDZMFeatures3D< T, R >::fillMatrix3D (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > distanceMap,
            boost::multi_array< double, 2 > & gldzmat )  [private]
```

In the method generateDistanceMap3D the distance map is generated, taking the matrix of the VOI as input. According to the position of a value in the matrix, the corresponding distance is saved in the distance map

Parameters

in		

10.17.2.2 getMatrix3D()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLDZMFeatures3D< T, R >::getMatrix3D (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > distanceMap )  [private]
```

In the method getMatrix3D the GLDZM matrix is generated and filled using the function fillMatrix. The function is mainly used get the size of the GLDZM matrix.

Parameters

in		

The documentation for this class was generated from the following file:

- GLDZMFeatures3D.h

10.18 GLRLMFeatures< T, R > Class Template Reference

```
#include <GLRLMFeatures.h>
```

Inheritance diagram for GLRLMFeatures< T, R >:

Public Member Functions

- void **defineGLRLMFeatures** (vector< string > &features)
- void getXYDirections (int &directionX, int &directionY, int angle)

  getXYDirections
- vector< double > calculateRowSums (boost::multi_array< double, 2 > glrlmatrix)

  calculateRowSums
- vector< double > calculateColSums (boost::multi_array< double, 2 > glrlmatrix)

  calculateColSums
- int **findIndex** (vector< T > array, int size, T target)
- void **getConfigValues** (ConfigFile config)
- void **setEmphasisValues** (int extEmph, double powRow, double powCol)
- double calculateTotalSum (boost::multi_array< double, 2 > glrlMatrix)
- int getMaxRunLength (boost::multi_array< T, R > inputMatrix)

  getMaxRunLength
- void calculateShortRunEmphasis (vector< double > colSums, double totalSum)

calculateShortRunEmphasis

- void calculateLongRunEmphasis (vector< double > colSums, double totalSum)

  calculateLongRunEmphasis
- void calculateLowGreyEmph (vector< double > colSums, double totalSum)

  calculateLowGreyEmph
- void calculateHighGreyEmph (vector< double > colSums, double totalSum)

  calculateHighGreyEmph
- void calculateShortRunLow (boost::multi_array< double, 2 > glrlmatrix, double totalSum)

  calculateShortRunLow
- void calculateShortRunHigh (boost::multi_array< double, 2 > glrlmatrix, double totalSum)

  calculateShortRunHigh
- void calculateLongRunLowEmph (boost::multi_array< double, 2 > glrlmatrix, double totalSum)

  calculateLongRunLowEmph
- void calculateLongRunHighEmph (boost::multi_array< double, 2 > glrlmatrix, double totalSum)

  calculateLongRunHighEmph
- void calculateGreyNonUniformity (vector< double > colSums, double totalSum)

  calculateGreyNonUniformity
- void calculateGreyNonUniformityNorm (vector< double > colSums, double totalSum)

  calculateGreyNonUniformityNorm
- void calculateRunLengthNonUniformityNorm (vector< double > rowSums, double totalSum)

  calculateRunLengthNonUniformityNorm
- void calculateRunLengthNonUniformity (vector< double > rowSums, double totalSum)

  calculateRunLengthNonUniformity
- void calculateTotalNrVoxels (boost::multi_array< T, R > inputMatrix, int depth)

  calculateTotalNrVoxels
- void **calculateTotalNrVoxels3D** (vector< T > vectorMatrElement)
- void calculateRunPercentage (boost::multi_array< T, R > inputMatrix, int depth, double totalSum, int nr↩ Neighbor)

  calculateRunPercentage
- void **calculateRunPercentage3D** (vector< T > vectorMatrElement, double totalSum, int nrNeighbor)
- boost::multi_array< double, 2 > **calculateProbMatrix** (boost::multi_array< double, 2 > glrlmatrix, double totalSum)
- double calculateMeanProbGrey (boost::multi_array< double, 2 > probMatrix)

  calculateMeanProbGrey
- void calculateGreyLevelVar (boost::multi_array< double, 2 > probMatrix, double mean)

  calculateGreyLevelVar
- double **calculateMeanProbRun** (boost::multi_array< double, 2 > probMatrix)
- void calculateRunLengthVar (boost::multi_array< double, 2 > probMatrix, double meanRun)

  calculateRunLengthVar
- void calculateRunEntropy (boost::multi_array< double, 2 > probMatrix)

  calculateRunEntropy

Public Attributes

- int **maxRunLength**
- float **powRow**
- float **powCol**
- int **calculateExtEmph**
- vector< T > **diffGreyLevels**
- double **shortRunEmphasis**
- double **longRunEmphasis**

- double **lowGreyEmph**
- double **highGreyEmph**
- double **shortRunLow**
- double **shortRunHigh**
- double **longRunLowEmph**
- double **longRunHighEmph**
- double **greyNonUniformity**
- double **greyNonUniformityNorm**
- double **runLengthNonUniformity**
- double **runLengthNonUniformityNorm**
- double **runPercentage**
- double **greyLevelVar**
- double **runLengthVar**
- double **runEntropy**

Private Types

- typedef boost::multi_array< double, 2 > **mat**

Private Member Functions

- void **extractGLRLMData** (vector< T > &glrlmData, GLRLMFeatures< T, R > glrlmFeatures)

Private Attributes

- double **totalSum**
- int **totalNrVoxels**

10.18.1 Detailed Description

template<class T, size_t R = 3>
class GLRLMFeatures< T, R >

The Grey Level Run Length Matrices also define a set of textural features.
The GLRL-matrices also investigate the distribution of the grey levels in the image. In this matrices the run length of a grey level in a direction Δ is counted.
Let N_g be the number of discretized grey levels present in the image.
Let N_r be the maximal run length present in the image.
Let M_Δ be the $N_g \times N_r$ GLRM matrix of direction Δ
In this matrix every element represents how often an intensity element i occurred j-times consecutively.
The directions in which the run length are counted are the same as for the GLCM matrices. The row number of the GLRLM matrix is representing the intensity value of the voxels and the column number is representing the run length of these voxels.
E.g. the matrix element $r_{ij} = r(i,j)$ is the number of occurences where discretized grey levels occur j-times consecutively.
Let $r_{i.} = \sum_{j=i}^{N_r} r_{ij}$ be the marginal sum of the runs of the run lengths j for grey level i
Let $r_{.j} = \sum_{i=1}^{N_g} r_{ij}$ be the marginal sum of the runs over the grey levels i for run length j
The feature values are calculated after calculating the GLRM matrices.
Also here are several methods possible to merge the matrices while calculating the features. The methods are the same as with the GLCM matrices.

10.18.2 Member Function Documentation

10.18.2.1 calculateColSums()

```
template<class T , size_t R>
vector< double > GLRLMFeatures< T, R >::calculateColSums (
            boost::multi_array< double, 2 > glrlmatrix )
```

calculateColSums

Parameters

boost::multi_array<double,2>	glrlmatrix : GLRM matrix

calculates the sum of columns and stores them in the vector colSums

10.18.2.2 calculateGreyLevelVar()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateGreyLevelVar (
            boost::multi_array< double, 2 > probMatrix,
            double meanGrey )
```

calculateGreyLevelVar

Parameters

boost::multi_array<double,2>	probMatrix : probability matrix
double	meanGrey : mean value of the grey levels

calculates the variance of grey levels
the lower the value, the more homogeneous is the region

10.18.2.3 calculateGreyNonUniformity()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateGreyNonUniformity (
            vector< double > colSums,
            double totalSum )
```

calculateGreyNonUniformity

Parameters

vector<double>	colSums : vector of the column sums
double	totalSum : sum of all matrix elements

This features is a measure for the distribution of the grey levels in the image matrix.
The more equally distrbuted the runs of the grey levels are, the lower is the value.

10.18.2.4 calculateGreyNonUniformityNorm()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateGreyNonUniformityNorm (
            vector< double > colSums,
            double totalSum )
```

calculateGreyNonUniformityNorm

Parameters

vector<double>	colSums : vector of the column sums
double	totalSum : sum of all matrix elements

This features is a normalized version of the grey-non-uniformity feature.

10.18.2.5 calculateHighGreyEmph()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateHighGreyEmph (
            vector< double > colSums,
            double totalSum )
```

calculateHighGreyEmph

Parameters

vector<double>	colSums : vector of the sums of the columns
double	totalSum : sum of all matrix elements

This feature emphasizes the high grey levels. The higher the value, the more high grey levels are in the matrix.

10.18.2.6 calculateLongRunEmphasis()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateLongRunEmphasis (
            vector< double > rowSums,
            double totalSum )
```

calculateLongRunEmphasis

Parameters

vector<double>	rowSums : vector of the sums of the rows
double	totalSum : sum of all matrix elements

This feature emphasizes the long runs. The higher the value, the more long runs are in the matrix.

10.18.2.7 calculateLongRunHighEmph()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateLongRunHighEmph (
            boost::multi_array< double, 2 > glrlmatrix,
            double totalSum )
```

calculateLongRunHighEmph

Parameters

boost::multi_array<double,2>	glrlmatrix : GLCM matrix
double	totalSum : sum of all matrix elements

This feature emphasizes the high grey levels which habe a long run. The higher the value, the more high grey levels with long runs are in the matrix.

10.18.2.8 calculateLongRunLowEmph()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateLongRunLowEmph (
            boost::multi_array< double, 2 > glrlmatrix,
            double totalSum )
```

calculateLongRunLowEmph

Parameters

boost::multi_array<double,2>	glrlmatrix : GLCM matrix
double	totalSum : sum of all matrix elements

This feature emphasizes the low grey levels which habe a long run. The higher the value, the more low grey levels with long runs are in the matrix.

10.18.2.9 calculateLowGreyEmph()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateLowGreyEmph (
            vector< double > colSums,
            double totalSum )
```

calculateLowGreyEmph

Parameters

vector<double>	colSums : vector of the sums of the columns
double	totalSum : sum of all matrix elements

This feature emphasizes the low grey levels. The higher the value, the more low grey levels are in the matrix.

10.18.2.10 calculateMeanProbGrey()

```
template<class T , size_t R>
double GLRLMFeatures< T, R >::calculateMeanProbGrey (
            boost::multi_array< double, 2 > probMatrix )
```

calculateMeanProbGrey

Parameters

boost::multi_array<double,2>	probMatrix matrix filled with the probabilities

calculates the mean probability of the appearance of every grey level TODO change bordwers in for loop (prob↩ Matrix.shape())

10.18.2.11 calculateRowSums()

```
template<class T , size_t R>
vector< double > GLRLMFeatures< T, R >::calculateRowSums (
            boost::multi_array< double, 2 > glrlmatrix )
```

calculateRowSums

Parameters

boost::multi_array<double,2>	glrlmatrix : GLRM matrix

calculates the sum of rows and stores them in the vector rowSums

10.18.2.12 calculateRunEntropy()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateRunEntropy (
            boost::multi_array< double, 2 > probMatrix )
```

calculateRunEntropy

Parameters

boost::multi_array<double,2>	probMatrix : probability matrix

calculates the entropy of the probability matrix

10.18.2.13 calculateRunLengthNonUniformity()

```
template<class T , size_t R>
```

```
void GLRLMFeatures< T, R >::calculateRunLengthNonUniformity (
            vector< double > rowSums,
            double totalSum )
```

calculateRunLengthNonUniformity

Parameters

vector<double>	colSums : vector of the column sums
double	totalSum : sum of all matrix elements

This feature is a measurement for the distribution of the run length.
The lower this value is, the more equally the run length are distributed.

10.18.2.14 calculateRunLengthNonUniformityNorm()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateRunLengthNonUniformityNorm (
            vector< double > rowSums,
            double totalSum )
```

calculateRunLengthNonUniformityNorm

Parameters

vector<double>	colSums : vector of the column sums
double	totalSum : sum of all matrix elements

This is a normalised version of the run-length non uniformity feature.

10.18.2.15 calculateRunLengthVar()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateRunLengthVar (
            boost::multi_array< double, 2 > probMatrix,
            double meanRun )
```

calculateRunLengthVar

Parameters

boost::multi_array<double,2>	probMatrix : probability matrix
double	meanRun : mean value of the run length

calculates the variance of run length
the lower the value, the more homogeneous is the region

10.18.2.16 calculateRunPercentage()

```
template<class T , size_t R>
```

```
void GLRLMFeatures< T, R >::calculateRunPercentage (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            double totalSum,
            int nrNeighbor )
```

calculateRunPercentage

Parameters

boost::multi_array<double,2>	glrlmatrix : GLRLM matrix
double	totalSum : sum of all matrix elements

calculates the fraction of runs appearing in the matrix and potential runs

10.18.2.17 calculateShortRunEmphasis()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateShortRunEmphasis (
            vector< double > rowSums,
            double totalSum )
```

calculateShortRunEmphasis

Parameters

vector<double>	rowSums : vector of the sums of the rows
double	totalSum : sum of all matrix elements

This feature emphasizes the short runs. The higher the value, the more short runs are in the matrix.

10.18.2.18 calculateShortRunHigh()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateShortRunHigh (
            boost::multi_array< double, 2 > glrlmatrix,
            double totalSum )
```

calculateShortRunHigh

Parameters

boost::multi_array<double,2>	glrlmatrix : GLCM matrix
double	totalSum : sum of all matrix elements

This feature emphasizes the high grey levels which habe a short run. The higher the value, the more high grey levels with short runs are in the matrix.

10.18.2.19 calculateShortRunLow()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateShortRunLow (
            boost::multi_array< double, 2 > glrlmatrix,
            double totalSum )
```

calculateShortRunLow

Parameters

boost::multi_array<double,2>	glrlmatrix : GLCM matrix
double	totalSum : sum of all matrix elements

This feature emphasizes the low grey levels which habe a short run. The higher the value, the more low grey levels with short runs are in the matrix.

10.18.2.20 calculateTotalNrVoxels()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::calculateTotalNrVoxels (
            boost::multi_array< T, R > inputMatrix,
            int depth )
```

calculateTotalNrVoxels

Parameters

boost::multi_array<double,2>	glrlmatrix : GLRLM matrix

calculates the total number of voxels in the matrix

TODO check if this is right!

10.18.2.21 calculateTotalSum()

```
template<class T , size_t R>
double GLRLMFeatures< T, R >::calculateTotalSum (
            boost::multi_array< double, 2 > glrlmatrix )
```

calculate the sum of all matrix elements

10.18.2.22 getMaxRunLength()

```
template<class T , size_t R>
int GLRLMFeatures< T, R >::getMaxRunLength (
            boost::multi_array< T, R > inputMatrix )
```

getMaxRunLength

Parameters

boost::multi_array<T,R>	inputMatrix get the maximal run length The maximal run length is the maximal size of one dimension

10.18.2.23 getXYDirections()

```
template<class T , size_t R>
void GLRLMFeatures< T, R >::getXYDirections (
            int & directionX,
            int & directionY,
            int angle )
```

getXYDirections

Parameters

int	directionX
int	directionY
int	angle

The function gets directionX and directionY as reference. Depending on the angle value, the parameter are set:
angle == 180 : go one pixel/voxel in x-direction; no move in y-direction
angle == 90 : no move in x-direction; go one pixel/voxel in y-direction
angle == 45 : go one pixel/voxel in x-direction; go one pixel/voxel in y direction
angle == 135 : go minus one pixel/voxel in x-direction; one pixel/voxel in y direction

The documentation for this class was generated from the following file:

- GLRLMFeatures.h

10.19 GLRLMFeatures2DAVG< T, R > Class Template Reference

```
#include <GLRLMFeatures2DAVG.h>
```

Inheritance diagram for GLRLMFeatures2DAVG< T, R >:

Collaboration diagram for GLRLMFeatures2DAVG< T, R >:

Public Member Functions

- void **calculateAllGLRLMFeatures2DAVG** (GLRLMFeatures2DAVG< T, R > &glrlmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffFGrey, ConfigFile config)
- void **writeCSVFileGLRLM2DAVG** (GLRLMFeatures2DAVG< T, R > glrlmFeat, string outputFolder)
- void **writeOneFileGLRLM2DAVG** (GLRLMFeatures2DAVG< T, R > glrlmFeat, string outputFolder)

Private Types

- typedef boost::multi_array< double, 2 > **glrlmMat**

Private Member Functions

- boost::multi_array< double, 2 > createGLRLMatrixAVG (boost::multi_array< T, R > inputMatrix, int depth, int ang)
- void **extractGLRLMDataAVG** (vector< T > &glrlmData, GLRLMFeatures2DAVG< T, R > glrlmFeatures)
- void fill2DMatrices2DAVG (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glrl↩ Matrix, int depth, int ang)

Private Attributes

- GLRLMFeatures< T, R > **glrlm**
- GLRLMFeatures2DVMRG< T, R > **glrlm2DFullMerge**
- double **totalSum**
- int **directionX**
- int **directionY**
- int **maxRunLength**

10.19.1 Detailed Description

template<class T, size_t R = 3>
class GLRLMFeatures2DAVG< T, R >

The class GLCMFeatures2DWOMerge inherits from the matrix GLCMFeatures.
It does not merge the matrices before feature calculation.
For every slice a GLCMatrix is calculated and from every of this matrices all features are extracted.
Then the average value of all features is calculated.

10.19.2 Member Function Documentation

10.19.2.1 createGLRLMatrixAVG()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLRLMFeatures2DAVG< T, R >::createGLRLMatrixAVG (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            int ang )  [private]
```

In the method createGLRLMatrixW=Merge the GLRLM-matrix for given slice is calculated

Parameters

in		

10.19.2.2 fill2DMatrices2DAVG()

```
template<class T , size_t R>
void GLRLMFeatures2DAVG< T, R >::fill2DMatrices2DAVG (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glrlMatrix,
            int depth,
            int ang )  [private]
```

In the method fill2DMatrices2DWOMerge the matrix is filled for the given image slice and angle

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLRLMFeatures2DAVG.h

10.20 GLRLMFeatures2DFullMerge< T, R > Class Template Reference

`#include <GLRLMFeatures2DFullMerge.h>`

Inheritance diagram for GLRLMFeatures2DFullMerge< T, R >:

Collaboration diagram for GLRLMFeatures2DFullMerge< T, R >:

Public Member Functions

- void **calculateAllGLRLMFeatures2DFullMerge** (GLRLMFeatures2DFullMerge< T, R > &glrlmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< T > vectorMatrElem, vector< double > spacing, ConfigFile config)
- void **writeCSVFileGLRLM2DFullMerge** (GLRLMFeatures2DFullMerge< T, R > glrlmFeat, string output↩Folder)
- void **writeOneFileGLRLM2DFullMerge** (GLRLMFeatures2DFullMerge< T, R > glrlmFeat, string output↩Folder)
- void fill2DMatrices2DFullMerge (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glrlMatrix, int depth, int ang)

Public Attributes

- float **powRow**
- float **powCol**

Private Member Functions

- boost::multi_array< double, 2 > createGLRLMatrixFullMerge (boost::multi_array< T, R > inputMatrix, int depth, int ang)
- void **extractGLRLMDataFullMerge** (vector< T > &glrlmData, GLRLMFeatures2DFullMerge< T, R > glrlmFeatures)

Private Attributes

- GLRLMFeatures< T, R > **glrlm**
- double **totalSum**
- int **directionX**
- int **directionY**
- int **maxRunLength**
- vector< double > **actualSpacing**
- string **normGLRLM**
- vector< double > **emphasisValues**

10.20.1 Detailed Description

template<class T, size_t R = 3>
class GLRLMFeatures2DFullMerge< T, R >

The class GLRLFeatures2DFullMerge herites from the class GLRLMFeatures.
It merges the matrices of all slices and calculates afterwards the features from the merged matrix.
All feature calculations are defined in the class GLRLMFeatures.
This class only contains the calculations of the merged matrix.

10.20.2 Member Function Documentation

10.20.2.1 createGLRLMatrixFullMerge()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLRLMFeatures2DFullMerge< T, R >::createGLRLMatrixFullMerge (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            int ang )  [private]
```

In the method createGLRLMatrixFullMerge the GLRLM-matrix for the given angle and a given slice is calculated

Parameters

in		

10.20.2.2 fill2DMatrices2DFullMerge()

```
template<class T , size_t R>
void GLRLMFeatures2DFullMerge< T, R >::fill2DMatrices2DFullMerge (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glrlMatrix,
            int depth,
            int ang )
```

In the method fill2DMatrices2DFullMerge the matrix is filled for the given image slice and angle

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLRLMFeatures2DFullMerge.h

10.21 GLRLMFeatures2DMRG< T, R > Class Template Reference

`#include <GLRLMFeatures2DMRG.h>`

Inheritance diagram for GLRLMFeatures2DMRG< T, R >:

Collaboration diagram for GLRLMFeatures2DMRG< T, R >:

Public Member Functions

- void **calculateAllGLRLMFeatures2DMRG** (GLRLMFeatures2DMRG< T, R > &glrlmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< double > spacing, ConfigFile config)
- void **writeCSVFileGLRLM2DMRG** (GLRLMFeatures2DMRG< T, R > glrlmFeat, string outputFolder)
- void **writeOneFileGLRLM2DMRG** (GLRLMFeatures2DMRG< T, R > glrlmFeat, string outputFolder)

Private Types

- typedef boost::multi_array< double, 2 > **glrlmMat**

Private Member Functions

- boost::multi_array< double, 2 > createGLRLMatrixMRG (boost::multi_array< T, R > inputMatrix, int depth)
- void **extractGLRLMDataMRG** (vector< T > &glrlmData, GLRLMFeatures2DMRG< T, R > glrlmFeatures)
- void fill2DMatrices2DMRG (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glrl↩ Matrix, int depth, int ang)

Private Attributes

- GLRLMFeatures< T, R > **glrlm**
- double **totalSum**
- int **directionX**
- int **directionY**
- int **maxRunLength**
- vector< double > **actualSpacing**
- string **normGLRLM**

10.21.1 Detailed Description

template<class T, size_t R = 3>
class GLRLMFeatures2DMRG< T, R >

The class GLRLMFeatures2DMRG herites from the class GLRLMFeatures.
It merges the matrices of every slice separately and calculates afterwards the features from the merged matrix.
Afterwards the mean value of the features for every slice is calculated.
The difference between this class and the other GLRLMFeature-classes is only the type of merging of the matrix.
All feature calculations are defined in the class GLRLMFeatures.
This class only contains the calculations of the merged matrix.

10.21.2 Member Function Documentation

10.21.2.1 createGLRLMatrixMRG()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLRLMFeatures2DMRG< T, R >::createGLRLMatrixMRG (
            boost::multi_array< T, R > inputMatrix,
            int depth )  [private]
```

In the method createGLRLMatrixMRG the GLRLM-matrix for given slice is calculated

Parameters

in		

10.21.2.2 fill2DMatrices2DMRG()

```
template<class T , size_t R>
void GLRLMFeatures2DMRG< T, R >::fill2DMatrices2DMRG (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glrlMatrix,
            int depth,
            int ang )  [private]
```

In the method fill2DMatrices2DMRG the matrix is filled for the given image slice and angle

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLRLMFeatures2DMRG.h

10.22 GLRLMFeatures2DVMRG< T, R > Class Template Reference

`#include <GLRLMFeatures2DVMRG.h>`

Inheritance diagram for GLRLMFeatures2DVMRG< T, R >:

Collaboration diagram for GLRLMFeatures2DVMRG< T, R >:

Public Member Functions

- void **calculateAllGLRLMFeatures2DVMRG** (GLRLMFeatures2DVMRG< T, R > &glrlmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< T > vectorMatrElem, vector< double > spacing, ConfigFile config)
- void **writeCSVFileGLRLM2DVMRG** (GLRLMFeatures2DVMRG< T, R > glrlmFeat, string outputFolder)
- void **writeOneFileGLRLM2DVMRG** (GLRLMFeatures2DVMRG< T, R > glrlmFeat, string outputFolder)
- void fill2DMatrices2DVMRG (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glrlMatrix, int depth, int ang)

Public Attributes

- float **powRow**
- float **powCol**

Private Member Functions

- boost::multi_array< double, 2 > createGLRLMatrixVMRG (boost::multi_array< T, R > inputMatrix, int depth, int ang)
- void **extractGLRLMDataVMRG** (vector< T > &glrlmData, GLRLMFeatures2DVMRG< T, R > glrlmFeatures)

Private Attributes

- GLRLMFeatures< T, R > **glrlm**
- double **totalSum**
- int **directionX**
- int **directionY**
- int **maxRunLength**
- vector< double > **actualSpacing**
- string **normGLRLM**
- vector< double > **emphasisValues**

10.22.1 Detailed Description

template<class T, size_t R = 3>
class GLRLMFeatures2DVMRG< T, R >

The class GLRLFeatures2DFullMerge herites from the class GLRLMFeatures.
It merges the matrices of all slices and calculates afterwards the features from the merged matrix.
All feature calculations are defined in the class GLRLMFeatures.
This class only contains the calculations of the merged matrix.

10.22.2 Member Function Documentation

10.22.2.1 createGLRLMatrixVMRG()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLRLMFeatures2DVMRG< T, R >::createGLRLMatrixVMRG (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            int ang )  [private]
```

In the method createGLRLMatrixFullMerge the GLRLM-matrix for the given angle and a given slice is calculated

Parameters

in		

10.22.2.2 fill2DMatrices2DVMRG()

```
template<class T , size_t R>
void GLRLMFeatures2DVMRG< T, R >::fill2DMatrices2DVMRG (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glrlMatrix,
```

```
int depth,
int ang )
```

In the method fill2DMatrices2DFullMerge the matrix is filled for the given image slice and angle

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLRLMFeatures2DVMRG.h

10.23 GLRLMFeatures2DWMerge< T, R > Class Template Reference

`#include <GLRLMFeatures2DWMerge.h>`

Inheritance diagram for GLRLMFeatures2DWMerge< T, R >:

Collaboration diagram for GLRLMFeatures2DWMerge< T, R >:

Public Member Functions

- void **calculateAllGLRLMFeatures2DWMerge** (GLRLMFeatures2DWMerge< T, R > &glrlmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< double > spacing, ConfigFile config)
- void **writeCSVFileGLRLM2DWMerge** (GLRLMFeatures2DWMerge< T, R > glrlmFeat, string outputFolder)
- void **writeOneFileGLRLM2DWMerge** (GLRLMFeatures2DWMerge< T, R > glrlmFeat, string outputFolder)

Private Types

- typedef boost::multi_array< double, 2 > **glrlmMat**

Private Member Functions

- boost::multi_array< double, 2 > createGLRLMatrixWMerge (boost::multi_array< T, R > inputMatrix, int depth)
- void **extractGLRLMDataWMerge** (vector< T > &glrlmData, GLRLMFeatures2DWMerge< T, R > glrlm↩Features)
- void fill2DMatrices2DWMerge (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glrlMatrix, int depth, int ang)

Private Attributes

- GLRLMFeatures< T, R > **glrlm**
- double **totalSum**
- int **directionX**
- int **directionY**
- int **maxRunLength**
- vector< double > **actualSpacing**
- string **normGLRLM**

10.23.1 Detailed Description

template<class T, size_t R = 3>
class GLRLMFeatures2DWMerge< T, R >

The class GLRLMFeatures2DWMerge herites from the class GLRLMFeatures.
It merges the matrices of every slice separately and calculates afterwards the features from the merged matrix.
Afterwards the mean value of the features for every slice is calculated.
The difference between this class and the other GLRLMFeature-classes is only the type of merging of the matrix.
All feature calculations are defined in the class GLRLMFeatures.
This class only contains the calculations of the merged matrix.

10.23.2 Member Function Documentation

10.23.2.1 createGLRLMatrixWMerge()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLRLMFeatures2DWMerge< T, R >::createGLRLMatrixWMerge (
            boost::multi_array< T, R > inputMatrix,
            int depth )  [private]
```

In the method createGLRLMatrixWMerge the GLRLM-matrix for given slice is calculated

Parameters

in		

10.23.2.2 fill2DMatrices2DWMerge()

```
template<class T , size_t R>
void GLRLMFeatures2DWMerge< T, R >::fill2DMatrices2DWMerge (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glrlMatrix,
            int depth,
            int ang )  [private]
```

In the method fill2DMatrices2DWMerge the matrix is filled for the given image slice and angle

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLRLMFeatures2DWMerge.h

10.24 GLRLMFeatures2DWOMerge< T, R > Class Template Reference

`#include <GLRLMFeatures2DWOMerge.h>`

Inheritance diagram for GLRLMFeatures2DWOMerge< T, R >:

Collaboration diagram for GLRLMFeatures2DWOMerge< T, R >:

Public Member Functions

- void **calculateAllGLRLMFeatures2DWOMerge** (GLRLMFeatures2DWOMerge< T, R > &glrlmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffFGrey, ConfigFile config)
- void **writeCSVFileGLRLM2DWOMerge** (GLRLMFeatures2DWOMerge< T, R > glrlmFeat, string output↩ Folder)
- void **writeOneFileGLRLM2DWOMerge** (GLRLMFeatures2DWOMerge< T, R > glrlmFeat, string output↩ Folder)

Private Types

- typedef boost::multi_array< double, 2 > **glrlmMat**

Private Member Functions

- boost::multi_array< double, 2 > createGLRLMatrixWOMerge (boost::multi_array< T, R > inputMatrix, int depth, int ang)
- void **extractGLRLMDataWOMerge** (vector< T > &glrlmData, GLRLMFeatures2DWOMerge< T, R > glrlmFeatures)
- void fill2DMatrices2DWOMerge (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glrlMatrix, int depth, int ang)

Private Attributes

- GLRLMFeatures< T, R > **glrlm**
- GLRLMFeatures2DFullMerge< T, R > **glrlm2DFullMerge**
- double **totalSum**
- int **directionX**
- int **directionY**
- int **maxRunLength**

10.24.1 Detailed Description

template<class T, size_t R = 3>
class GLRLMFeatures2DWOMerge< T, R >

The class GLCMFeatures2DWOMerge inherits from the matrix GLCMFeatures.
It does not merge the matrices before feature calculation.
For every slice a GLCMatrix is calculated and from every of this matrices all features are extracted.
Then the average value of all features is calculated.

10.24.2 Member Function Documentation

10.24.2.1 createGLRLMatrixWOMerge()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLRLMFeatures2DWOMerge< T, R >::createGLRLMatrixWOMerge (
            boost::multi_array< T, R > inputMatrix,
            int depth,
            int ang )  [private]
```

In the method createGLRLMatrixW=Merge the GLRLM-matrix for given slice is calculated

Parameters

in		

10.24.2.2 fill2DMatrices2DWOMerge()

```
template<class T , size_t R>
void GLRLMFeatures2DWOMerge< T, R >::fill2DMatrices2DWOMerge (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glrlMatrix,
            int depth,
            int ang )  [private]
```

In the method fill2DMatrices2DWOMerge the matrix is filled for the given image slice and angle

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

The documentation for this class was generated from the following file:

- GLRLMFeatures2DWOMerge.h

10.25 GLRLMFeatures3D< T, R > Class Template Reference

```
#include <GLRLMFeatures3D.h>
```

Inheritance diagram for GLRLMFeatures3D< T, R >:

Collaboration diagram for GLRLMFeatures3D< T, R >:

Public Member Functions

- void **calculateAllGLRLMFeatures3D** (GLRLMFeatures3D< T, R > &glrlmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< T > vectorMatrElem, vector< double > spacing, ConfigFile config)
- void **writeCSVFileGLRLM3D** (GLRLMFeatures3D< T, R > glrlmFeat, string outputFolder)
- void **writeOneFileGLRLM3D** (GLRLMFeatures3D< T, R > glrlmFeat, string outputFolder)

Private Member Functions

- boost::multi_array< double, 2 > createGLRLMatrix3D (boost::multi_array< T, R > inputMatrix)
- void **extractGLRLMData3D** (vector< T > &glrlmData, GLRLMFeatures3D< T, R > glrlmFeatures)
- void fill3DMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glrlMatrix, int directionX, int directionY, int directionZ)

Private Attributes

- GLRLMFeatures< T, R > **glrlm**
- GLRLMFeatures3DAVG< T, R > **glrlm3D**
- vector< double > **actualSpacing**
- string **normGLRLM**
- double **totalSum**
- int **maxRunLength**
- int **totalNrVoxels**

10.25.1 Detailed Description

template<class T, size_t R = 3>
class GLRLMFeatures3D< T, R >

The class GLRLMFeatures3D herites from the class GLRLMFeatures.
To calculate the run length it goes also in the depth.
It calculates the feature values for every angle and calculates then the mean value of the features.
All feature calculations are defined in the class GLRLMFeatures.

10.25.2 Member Function Documentation

10.25.2.1 createGLRLMatrix3D()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLRLMFeatures3D< T, R >::createGLRLMatrix3D (
            boost::multi_array< T, R > inputMatrix )  [private]
```

The method createGLRLMatrix3D sums up all matrices of the different directions

Parameters

in	*inputMatrix*	the original matrix of the VOI

10.25.2.2 fill3DMatrices()

```
template<class T , size_t R>
void GLRLMFeatures3D< T, R >::fill3DMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & glrlMatrix,
            int directionX,
            int directionY,
            int directionZ )  [private]
```

The method getNeighbors3D stores all neighbor pairs for the desired angle and the actual input matrix in a vector

Parameters

in	*inputMatrix*	the original matrix of the VOI
in	*directionX*	
in	*directionY*	
in	*directionZ*	The direction-parameters determine in which direction the run length is calculated

The documentation for this class was generated from the following file:

- GLRLMFeatures3D.h

10.26 GLRLMFeatures3DAVG< T, R > Class Template Reference

`#include <GLRLMFeatures3DAVG.h>`

Inheritance diagram for GLRLMFeatures3DAVG< T, R >:

Collaboration diagram for GLRLMFeatures3DAVG< T, R >:

Public Member Functions

- void **calculateAllGLRLMFeatures3DAVG** (GLRLMFeatures3DAVG< T, R > &glrlmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< T > vectorMatrElem, ConfigFile config)
- void **writeCSVFileGLRLM3DAVG** (GLRLMFeatures3DAVG< T, R > glrlmFeat, string outputFolder)
- void **writeOneFileGLRLM3DAVG** (GLRLMFeatures3DAVG< T, R > glrlmFeat, string outputFolder)
- void **getXYdirections3D** (int &directionX, int &directionY, int &directionZ, int ang)

Private Member Functions

- boost::multi_array< double, 2 > **createGLRLMatrix3D** (boost::multi_array< T, R > inputMatrix, int ang)
- void **extractGLRLMData3D** (vector< T > &glrlmData, GLRLMFeatures3DAVG< T, R > glrlmFeatures)
- void **fill3DMatrices** (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glrlMatrix, int directionX, int directionY, int directionZ)
- void **getMaxRunLength3D** (boost::multi_array< T, R > inputMatrix)

Private Attributes

- GLRLMFeatures< T, R > **glrlm**
- double **totalSum**
- int **maxRunLength**
- int **totalNrVoxels**

10.26.1 Detailed Description

template<class T, size_t R = 3>
class GLRLMFeatures3DAVG< T, R >

The class GLRLMFeatures3DAVG herites from the class GLRLMFeatures.
A GLRLM matrix is calculated for every angle separately. From all these matrices the feature values are then calculated and the mean value is calculated.
All feature calculations are defined in the class GLRLMFeatures.

The documentation for this class was generated from the following file:

- GLRLMFeatures3DAVG.h

10.27 GLRLMFeatures3DWOMerge< T, R > Class Template Reference

`#include <GLRLMFeatures3DWOMerge.h>`

Inheritance diagram for GLRLMFeatures3DWOMerge< T, R >:

Collaboration diagram for GLRLMFeatures3DWOMerge< T, R >:

Public Member Functions

- void **calculateAllGLRLMFeatures3DWOMerge** (GLRLMFeatures3DWOMerge< T, R > &glrlmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< T > vectorMatrElem, ConfigFile config)
- void **writeCSVFileGLRLM3DWOMerge** (GLRLMFeatures3DWOMerge< T, R > glrlmFeat, string output↩ Folder)
- void **writeOneFileGLRLM3DWOMerge** (GLRLMFeatures3DWOMerge< T, R > glrlmFeat, string output↩ Folder)
- void **getXYdirections3D** (int &directionX, int &directionY, int &directionZ, int ang)

Private Member Functions

- boost::multi_array< double, 2 > **createGLRLMatrix3D** (boost::multi_array< T, R > inputMatrix, int ang)
- void **extractGLRLMData3D** (vector< T > &glrlmData, GLRLMFeatures3DWOMerge< T, R > glrlm↩ Features)
- void **fill3DMatrices** (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glrlMatrix, int directionX, int directionY, int directionZ)
- void **getMaxRunLength3D** (boost::multi_array< T, R > inputMatrix)

Private Attributes

- GLRLMFeatures< T, R > **glrlm**
- double **totalSum**
- int **maxRunLength**
- int **totalNrVoxels**

10.27.1 Detailed Description

template<class T, size_t R = 3>
class GLRLMFeatures3DWOMerge< T, R >

The class GLRLMFeatures3DWOMerge herites from the class GLRLMFeatures.
A GLRLM matrix is calculated for every angle separately. From all these matrices the feature values are then calculated and the mean value is calculated.
All feature calculations are defined in the class GLRLMFeatures.

The documentation for this class was generated from the following file:

- GLRLMFeatures3DWOMerge.h

10.28 GLSZMFeatures2DAVG< T, R > Class Template Reference

`#include <GLSZMFeatures2DAVG.h>`

Inheritance diagram for GLSZMFeatures2DAVG< T, R >:

Collaboration diagram for GLSZMFeatures2DAVG< T, R >:

Public Member Functions

- void getNeighbors (boost::multi_array< T, R > &inputMatrix, T actElement, vector< vector< int > > &matrixIndices)
- void **getALLXYDirections** (int &directionX, int &directionY, int angle)
- void **calculateAllGLSZMFeatures2DAVG** (GLSZMFeatures2DAVG< T, R > &GLSZMFeat, boost::multi_↩ array< T, R > inputMatrix, vector< T > diffGrey, ConfigFile config)
- void **writeCSVFileGLSZM2DAVG** (GLSZMFeatures2DAVG< T, R > GLSZMFeat, string outputFolder)
- void **writeOneFileGLSZM2DAVG** (GLSZMFeatures2DAVG< T, R > GLSZMFeat, string outputFolder)

Private Member Functions

- void **extractGLSZMData** (vector< T > &GLSZMData, GLSZMFeatures2DAVG< T, R > GLSZMFeatures)
- void fill2DGLSZMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glc↩ Matrix, int depth)
- boost::multi_array< double, 2 > getGLSZMMatrix (boost::multi_array< T, R > inputMatrix, int depth)
- int **getBiggestZoneNr** (boost::multi_array< T, R > inputMatrix)

Private Attributes

- GLSZMFeatures2DMRG< T, R > **glszm2D**
- GLRLMFeatures< T, R > **glrlm**
- int **maxZoneSize**
- vector< double > **rowSums**
- vector< double > **colSums**

Additional Inherited Members

10.28.1 Detailed Description

template<class T, size_t R>
class GLSZMFeatures2DAVG< T, R >

The class GLSZMFeatures2DAVG herites from the class GLSZMFeatures2D, because the feature calculations are the same. Only the matrix calculation is different
All feature calculations are defined in the class GLRLMFeatures.
This class calculates a GLSZM matrix for every slice of the VOI and extracts the feature values from every slice. Then a mean value of all these feature values is calculated.

For grey level size zone matrices, groups of connected voxels with a specific grey value and size are grouped. A voxel is connected with another voxel if they have the same grey level.

10.28.2 Member Function Documentation

10.28.2.1 fill2DGLSZMatrices()

```
template<class T , size_t R>
void GLSZMFeatures2DAVG< T, R >::fill2DGLSZMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & GLSZMatrix,
            int depth )  [private]
```

In the method fill2DGLSZMatrices the GLSZM matrices are filled using the matrix filled with the intensity values of the VOI.

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.28.2.2 getGLSZMMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLSZMFeatures2DAVG< T, R >::getGLSZMMatrix (
            boost::multi_array< T, R > inputMatrix,
            int depth )  [private]
```

In the method getGLSZMMatrix the GLSZM matrices with the right size are generated and filled using the fill2DG↩ LSZM function.

Parameters

in	*inputMatrix*	the original matrix of the VOI
in	*depth*	number of actual slice for which the GLSZM should be calculated
out		

10.28.2.3 getNeighbors()

```
template<class T , size_t R>
void GLSZMFeatures2DAVG< T, R >::getNeighbors (
            boost::multi_array< T, R > & inputMatrix,
            T actElement,
            vector< vector< int > > & matrixIndices )
```

In the method getNeighbors the number of voxels in the biggest zone is determined

Parameters

in	*inputMatrix*	the original matrix of the VOI
out		

The documentation for this class was generated from the following file:

- GLSZMFeatures2DAVG.h

10.29 GLSZMFeatures2DMRG< T, R > Class Template Reference

`#include <GLSZMFeatures2D.h>`

Inheritance diagram for GLSZMFeatures2DMRG< T, R >:

Collaboration diagram for GLSZMFeatures2DMRG< T, R >:

Public Member Functions

- void **defineGLSZMFeatures** (vector< string > &features)
- void getNeighbors (boost::multi_array< T, R > &inputMatrix, T actElement, vector< vector< int > > &matrixIndices)
- void **getALLXYDirections** (int &directionX, int &directionY, int angle)
- void **calculateAllGLSZMFeatures2DMRG** (GLSZMFeatures2DMRG< T, R > &GLSZMFeat, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< T > vectorMatrElem, ConfigFile config)
- void **writeCSVFileGLSZM** (GLSZMFeatures2DMRG< T, R > GLSZMFeat, string outputFolder)
- void **writeOneFileGLSZM** (GLSZMFeatures2DMRG< T, R > GLSZMFeat, string outputFolder)

Private Member Functions

- int getBiggestZoneNr (boost::multi_array< T, R > inputMatrix)
- boost::multi_array< double, 2 > getGLSZMMatrix (boost::multi_array< T, R > inputMatrix)
- void fill2DGLSZMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glcMatrix)
- void **extractGLSZMData** (vector< T > &GLSZMData, GLSZMFeatures2DMRG< T, R > GLSZMFeatures)

Private Attributes

- int **maxZoneSize**
- vector< double > **rowSums**
- vector< double > **colSums**
- GLRLMFeatures< T, R > **glrlm**

Additional Inherited Members

10.29.1 Detailed Description

template<class T, size_t R>
class GLSZMFeatures2DMRG< T, R >

The class GLSZMFeatures2DMRG herites from the class GLRLMFeatures, because the feature calculations are the same. Only the matrix calculation is different
All feature calculations are defined in the class GLRLMFeatures.
This class only contains the calculations of the 2D matrix.

For grey level size zone matrices, groups of connected voxels with a specific grey value and size are grouped. A voxel is connected with another voxel if they have the same grey level.

10.29.2 Member Function Documentation

10.29.2.1 fill2DGLSZMatrices()

```
template<class T , size_t R>
void GLSZMFeatures2DMRG< T, R >::fill2DGLSZMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & GLSZMatrix )  [private]
```

In the method fill2DGLSZMatrices the GLSZM matrices are filled using the matrix filled with the intensity values of the VOI.

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.29.2.2 getBiggestZoneNr()

```
template<class T , size_t R>
int GLSZMFeatures2DMRG< T, R >::getBiggestZoneNr (
            boost::multi_array< T, R > inputMatrix )  [private]
```

In the method getBiggestZoneNr the number of voxels in the biggest zone is determined

Parameters

in	*inputMatrix*	the original matrix of the VOI
out		

10.29.2.3 getGLSZMMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLSZMFeatures2DMRG< T, R >::getGLSZMMatrix (
            boost::multi_array< T, R > inputMatrix )  [private]
```

In the method getGLSZMMatrix the GLSZM matrices with the right size are generated and filled using the fill2DG↩ LSZM function.

Parameters

in	*inputMatrix*	the original matrix of the VOI
out		

10.29.2.4 getNeighbors()

```
template<class T , size_t R>
void GLSZMFeatures2DMRG< T, R >::getNeighbors (
            boost::multi_array< T, R > & inputMatrix,
            T actElement,
            vector< vector< int > > & matrixIndices )
```

In the method getNeighbors the number of voxels in the biggest zone is determined

Parameters

in	*inputMatrix*	the original matrix of the VOI
out		

The documentation for this class was generated from the following file:

- GLSZMFeatures2D.h

10.30 GLSZMFeatures2DWOMerge< T, R > Class Template Reference

`#include <GLSZMFeatures2DWOMerge.h>`

Inheritance diagram for GLSZMFeatures2DWOMerge< T, R >:

Collaboration diagram for GLSZMFeatures2DWOMerge< T, R >:

Public Member Functions

- void getNeighbors (boost::multi_array< T, R > &inputMatrix, T actElement, vector< vector< int > > &matrixIndices)
- void **getALLXYDirections** (int &directionX, int &directionY, int angle)
- void **calculateAllGLSZMFeatures2DWOMerge** (GLSZMFeatures2DWOMerge< T, R > &GLSZMFeat, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, ConfigFile config)
- void **writeCSVFileGLSZM2DWOMerge** (GLSZMFeatures2DWOMerge< T, R > GLSZMFeat, string outputFolder)
- void **writeOneFileGLSZM2DWOMerge** (GLSZMFeatures2DWOMerge< T, R > GLSZMFeat, string outputFolder)

Private Member Functions

- void **extractGLSZMData** (vector< T > &GLSZMData, GLSZMFeatures2DWOMerge< T, R > GLSZM↩ Features)
- void fill2DGLSZMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glc↩ Matrix, int depth)
- boost::multi_array< double, 2 > getGLSZMMatrix (boost::multi_array< T, R > inputMatrix, int depth)
- int **getBiggestZoneNr** (boost::multi_array< T, R > inputMatrix)

Private Attributes

- GLSZMFeatures2D< T, R > **glszm2D**
- int **maxZoneSize**
- vector< double > **rowSums**
- vector< double > **colSums**

10.30.1 Detailed Description

template<class T, size_t R>
class GLSZMFeatures2DWOMerge< T, R >

The class GLSZMFeatures2DWOMerge herites from the class GLSZMFeatures2D, because the feature calculations are the same. Only the matrix calculation is different
All feature calculations are defined in the class GLRLMFeatures.
This class calculates a GLSZM matrix for every slice of the VOI and extracts the feature values from every slice. Then a mean value of all these feature values is calculated.

For grey level size zone matrices, groups of connected voxels with a specific grey value and size are grouped. A voxel is connected with another voxel if they have the same grey level.

10.30.2 Member Function Documentation

10.30.2.1 fill2DGLSZMatrices()

```
template<class T , size_t R>
void GLSZMFeatures2DWOMerge< T, R >::fill2DGLSZMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & GLSZMatrix,
            int depth )  [private]
```

In the method fill2DGLSZMatrices the GLSZM matrices are filled using the matrix filled with the intensity values of the VOI.

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.30.2.2 getGLSZMMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLSZMFeatures2DWOMerge< T, R >::getGLSZMMatrix (
            boost::multi_array< T, R > inputMatrix,
            int depth )  [private]
```

In the method getGLSZMMatrix the GLSZM matrices with the right size are generated and filled using the fill2DG↩LSZM function.

Parameters

in	*inputMatrix*	the original matrix of the VOI
in	*depth*	number of actual slice for which the GLSZM should be calculated
out		

10.30.2.3 getNeighbors()

```
template<class T , size_t R>
void GLSZMFeatures2DWOMerge< T, R >::getNeighbors (
            boost::multi_array< T, R > & inputMatrix,
            T actElement,
            vector< vector< int > > & matrixIndices )
```

In the method getNeighbors the number of voxels in the biggest zone is determined

Parameters

in	*inputMatrix*	the original matrix of the VOI
out		

The documentation for this class was generated from the following file:

- GLSZMFeatures2DWOMerge.h

10.31 GLSZMFeatures3D< T, R > Class Template Reference

`#include <GLSZMFeatures3D.h>`

Inheritance diagram for GLSZMFeatures3D< T, R >:

Collaboration diagram for GLSZMFeatures3D< T, R >:

Public Member Functions

- void getNeighbors3D (boost::multi_array< T, R > &inputMatrix, T actElement, vector< vector< int > > &matrixIndices)
- void **calculateAllGLSZMFeatures3D** (GLSZMFeatures3D< T, R > &GLSZMFeat, Image< T, R > image↩Attr, ConfigFile config)
- void **writeCSVFileGLSZM3D** (GLSZMFeatures3D< T, R > GLSZMFeat, string outputFolder)
- void **writeOneFileGLSZM3D** (GLSZMFeatures3D< T, R > GLSZMFeat, string outputFolder)

Private Member Functions

- void **extractGLSZMData3D** (vector< T > &GLSZMData, GLSZMFeatures3D< T, R > GLSZMFeatures)
- boost::multi_array< double, 2 > getGLSZMMatrix3D (boost::multi_array< T, R > inputMatrix, vector< T > vectorMatrElem)
- void fill3DGLSZMatrices (boost::multi_array< T, R > inputMatrix, boost::multi_array< double, 2 > &glc↩Matrix)
- int **getBiggestZoneNr3D** (vector< T > vectorMatrElem)

Private Attributes

- GLSZMFeatures2DMRG< T, R > **GLSZM2D**
- GLRLMFeatures< T, R > **glrlm**
- int **maxZoneSize**
- vector< double > **rowSums**
- vector< double > **colSums**

Additional Inherited Members

10.31.1 Detailed Description

template<class T, size_t R>
class GLSZMFeatures3D< T, R >

The class GLSZMFeatures3D herites from the class GLSZMFeatures2D, because the feature calculations are the same. Only the matrix calculation is different
All feature calculations are defined in the class GLRLMFeatures.
This class calculates a GLSZM matrix considering 13 neighbors(3D approach)

For grey level size zone matrices, groups of connected voxels with a specific grey value and size are grouped. A voxel is connected with another voxel if they have the same grey level.

10.31.2 Member Function Documentation

10.31.2.1 fill3DGLSZMatrices()

```
template<class T , size_t R>
void GLSZMFeatures3D< T, R >::fill3DGLSZMatrices (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< double, 2 > & GLSZMatrix )  [private]
```

In the method fill3DGLSZMatrices the GLSZM matrices are filled using the matrix filled with the intensity values of the VOI.

Parameters

in	*inputMatrix*	the original matrix of the VOI
in		

10.31.2.2 getGLSZMMatrix3D()

```
template<class T , size_t R>
boost::multi_array< double, 2 > GLSZMFeatures3D< T, R >::getGLSZMMatrix3D (
```

```
            boost::multi_array< T, R > inputMatrix,
            vector< T > vectorMatrElem )  [private]
```

In the method getGLSZMMatrix3D the GLSZM matrices with the right size are generated and filled using the fill2↩ DGLSZM function.

Parameters

in	*inputMatrix*	the original matrix of the VOI
in	*depth*	number of actual slice for which the GLSZM should be calculated
out		

10.31.2.3 getNeighbors3D()

```
template<class T , size_t R>
void GLSZMFeatures3D< T, R >::getNeighbors3D (
            boost::multi_array< T, R > & inputMatrix,
            T actElement,
            vector< vector< int > > & matrixIndices )
```

In the method getNeighbors3D the number of voxels in the biggest zone is determined

Parameters

in	*inputMatrix*	the original matrix of the VOI
out		

The documentation for this class was generated from the following file:

- GLSZMFeatures3D.h

10.32 Image< T, R > Class Template Reference

Public Member Functions

- **Image** (unsigned int row, unsigned int col, unsigned int depth)
- void getInterpolatedImageMask (ConfigFile config, ImageType *image, ImageType *mask)

  getInterpolatedImageMask
- ImageType::Pointer getResampledImage (ImageType *originalImage, double *outputSpacing, itk::Size< 3 > outputSize)

  getResampledImage resample the image to the desired outputSize with desired spacing
- boost::multi_array< T, R > get3Dimage (ImageType *image, ImageType *mask, ConfigFile config)

  get3Dimage
- boost::multi_array< T, R > get3DimageResegmented (ImageType *image, ImageType *mask, ConfigFile config)

  get3DimageResegmented
- boost::multi_array< T, R > get3DimageLocalInt (ImageType *image, ImageType *mask)

get3DimageLocalInt

- vector< T > getGreyLevels ()

  getGreyLevels() the fuction returns a vector, containing the different grey levels of the matrix

- vector< T > getVectorOfMatrixElementsNotNAN (boost::multi_array< T, R > inputMatrix)

  getVectorOfMatrixElementsNotNAN In this function, all matrix elements, which are contained in the mask, are stored in a vector

- void discretizationFixedWidth (boost::multi_array< T, R > &inputMatrix, double intervalWidth)

  discretizationFixedWidth

- void discretizationFixedBinNr (boost::multi_array< T, R > &inputMatrix, vector< T > elementVector, double binNr)

  discretizationFixedBinNr

- void calculateSUV (boost::multi_array< T, R > &inputMatrix, ConfigFile config)

  calculateSUV this function is only called if the image is a PET-image
  it calculates the SUV-values from the original image using the information given by the user

- void calculateSUL (boost::multi_array< T, R > &inputMatrix, ConfigFile config)

  calculateSUL this function is only called if the image is a PET-image
  it calculates the SUL-values from the original image using the information given by the user.

- void getImageAttributes (ImageType *filteredImage, ImageType *mask, ConfigFile configName)

  getImageAttributes

- void getImageAttributesDiscretized (ImageType *filteredImage, ImageType *maskFilter, ConfigFile config↩ Name)

  getImageAttributesDIscretized

- int getValueInMask (ImageType::Pointer mask)

Public Attributes

- vector< T > **diffGreyLevels**
- boost::multi_array< T, R > **imageMatrix**
- boost::multi_array< T, R > **imageMatrixOriginal**
- boost::multi_array< T, R > **imageMatrixLocalInt**
- boost::multi_array< T, R > **imageMatrixIVH**
- vector< T > **vectorOfMatrixElements**
- vector< T > **vectorOfMatrixElementsOriginal**
- int **nrRows**
- int **nrCols**
- int **nrDepth**
- T **minGreyLevel**
- T **maxGreyLevel**
- ImageType::Pointer **image**
- ImageType::Pointer **mask**

Private Attributes

- int **nrBins**
- double **binWidth**
- double **SUVfactor**
- int **maxValueInMask**

10.32.1 Member Function Documentation

10.32.1.1 discretizationFixedBinNr()

```
template<class T , size_t R>
void Image< T, R >::discretizationFixedBinNr (
             boost::multi_array< T, R > & inputMatrix,
             vector< T > elementVector,
             double binNr )
```

discretizationFixedBinNr

Parameters

inputMatrix	
binNr	Discretizes the intensity values inside the VOI to a fixed number of bins The number of bins can be set by the user in the .config file

10.32.1.2 discretizationFixedWidth()

```
template<class T , size_t R>
void Image< T, R >::discretizationFixedWidth (
             boost::multi_array< T, R > & inputMatrix,
             double intervalWidth )
```

discretizationFixedWidth

Parameters

inputMatrix	
intervalWidth	The intensity values of the VOI are discretized using a fixed bin size The bin size is given by the user in the .config file

10.32.1.3 get3Dimage()

```
template<class T , size_t R>
boost::multi_array< T, R > Image< T, R >::get3Dimage (
             ImageType * image,
             ImageType * mask,
             ConfigFile config )
```

get3Dimage

Parameters

in	*ImageType*	image: original image
in	*ImageType*	mask: original mask the image values are assigned to a boost::multi_array image matrix, if they are lying inside the mask

10.32.1.4 get3DimageLocalInt()

```
template<class T , size_t R>
boost::multi_array< T, R > Image< T, R >::get3DimageLocalInt (
            ImageType * image,
            ImageType * mask )
```

get3DimageLocalInt

Parameters

in	*ImageType*	image: original image
in	*ImageType*	mask: original mask the image values are assigned to a boost::multi_array image matrix, if they are lying in the bounding box. For the calculation of the local intensity features, not only the image values inside the mask are required, but also the values lying outside the mask.

10.32.1.5 get3DimageResegmented()

```
template<class T , size_t R>
boost::multi_array< T, R > Image< T, R >::get3DimageResegmented (
            ImageType * image,
            ImageType * mask,
            ConfigFile config )
```

get3DimageResegmented

Parameters

in	*ImageType*	image: original image
in	*ImageType*	mask: original mask the image values are assigned to a boost::multi_array image matrix, if they are lying inside the mask. Intensity values below or above the set resegmentation values are deleted from the mask.

10.32.1.6 getImageAttributes()

```
template<class T , size_t R>
void Image< T, R >::getImageAttributes (
            ImageType * filteredImage,
            ImageType * maskFilter,
            ConfigFile config )
```

getImageAttributes

Parameters

filteredImage	
maskFilter	The function getImageAttributes takes filtered image and mask and stores the image information in a multi-dimensional array (using the get3Dimage-function) It stores the elements which are inside the mask in an array - using the getVectorElementsNotNAN-function (in order to calculate the statistical features)

10.32.1.7 getImageAttributesDiscretized()

```
template<class T , size_t R>
void Image< T, R >::getImageAttributesDiscretized (
             ImageType * filteredImage,
             ImageType * maskFilter,
             ConfigFile config )
```

getImageAttributesDIscretized

Parameters

filteredImage	
maskFilter	The function getImageAttributesDiscretized is called before the textural features are calculated: It discretizes the image matrix using the discretization method set by the user - using the discretization function It stores the values of the discretized image in a matrix (using the get3Dimage-function) It stores the elements of this matrix which are inside the mask in an array (using getVectorOfMatrixElementsNotNAN)

10.32.1.8 getInterpolatedImageMask()

```
template<class T , size_t R>
void Image< T, R >::getInterpolatedImageMask (
             ConfigFile config,
             ImageType * imageFiltered,
             ImageType * maskFiltered )
```

getInterpolatedImageMask

The image and the mask are interpolated using the nearest neighbor algorithm using up- or downsampling, what was set by the user
In a later version, the user should be able to choose the interpolation method

10.32.1.9 getResampledImage()

```
template<class T , size_t R>
ImageType::Pointer Image< T, R >::getResampledImage (
             ImageType * originalImage,
             double * outputSpacing,
             itk::Size< 3 > outputSize )
```

getResampledImage resample the image to the desired outputSize with desired spacing

Parameters

in	*originalImage*	
in	*double*	outputSpacing: the spacing of the resampled image
in	*itk::size*	outputSize: the size of the resampled image

10.32.1.10 getValueInMask()

```
template<class T , size_t R>
int Image< T, R >::getValueInMask (
             ImageType::Pointer mask )
```

In the function getValueInMask the value inside the mask is detected.
It can vary from 1 - 100. The value inside the mask is needed in order to create a mesh from this mask.

Parameters

in		

The documentation for this class was generated from the following file:

- image.h

10.33 IntensityHistogram< T, R > Class Template Reference

`#include <intensityHistogram.h>`

Inheritance diagram for IntensityHistogram< T, R >:

Collaboration diagram for IntensityHistogram< T, R >:

Public Member Functions

- void **calculateAllIntFeatures** (IntensityHistogram< T, R > &intense, boost::multi_array< T, R > inputMatrix, vector< T > vectorOfMatrElements, vector< T > diffGrey)
- void **writeCSVFileIntensity** (IntensityHistogram< T, R > intensHist, string outputFolder)
- void **writeOneFileIntensity** (IntensityHistogram< T, R > intense, string outputFolder)

Private Types

- typedef accumulator_set< T, features< tag::density > > **accIntensity**

Private Member Functions

- void getProbabilities (boost::multi_array< T, R > inputMatrix)
- void **getHistGradient** ()
- void **extractIntenseData** (vector< T > &intenseData, IntensityHistogram< T, R > intenseFeatures)
- void getNrElements (vector< double > &nrElements)
- void getHistUniformity ()
- void getEntropy ()
- void getMode ()
- void **getMaxHistGradient** ()
- void **getMinHistGradient** ()
- void **defineIntenseFeatures** (vector< string > &features)

Private Attributes

- vector< T > **probabilities**
- T **entropy**
- T **mode**
- T **histUniformity**
- T **maxHistGradient**
- T **maxHistGradGreyValue**
- T **minHistGradient**
- T **minHistGradGreyValue**
- vector< T > **vectorOfMatrixElem**
- vector< double > **nrElementsH**
- vector< T > **diffGreyLevels**
- vector< double > **probElements**
- vector< T > **maxHistVecGradient**
- vector< T > **minHistVecGradient**

10.33.1 Detailed Description

template<class T, size_t R>
class IntensityHistogram< T, R >

In the class IntensityHistogram the intensity histogram features are calculated.
The class inherits from the class StatisticalFeatures, as the majority of the features are the same.
The calculation of the feature values is done after discretizing the matrix values to a user specified bin number.

10.33.2 Member Function Documentation

10.33.2.1 getEntropy()

```
template<class T , size_t R>
void IntensityHistogram< T, R >::getEntropy ( )  [private]
```

The method getEntropy calculates the entropy of the probabilities

10.33.2.2 getHistUniformity()

```
template<class T , size_t R>
void IntensityHistogram< T, R >::getHistUniformity ( )  [private]
```

The method getHistUniformity calculates the uniformity of the histogram of the discretized grey levels.

10.33.2.3 getMode()

```
template<class T , size_t R>
void IntensityHistogram< T, R >::getMode ( )  [private]
```

The method getMode calculates the mode of the distribution.

Parameters

in	*inputMatrix*	the original matrix of the VOI

10.33.2.4 getNrElements()

```
template<class T , size_t R>
void IntensityHistogram< T, R >::getNrElements (
            vector< double > & nrElements )  [private]
```

The method getNrElements gets for every grey level the amount of voxel with the specific grey level.

Parameters

in	*vector*	nrElements: reference to a vector, where the number of different elements are stored

10.33.2.5 getProbabilities()

```
template<class T , size_t R>
void IntensityHistogram< T, R >::getProbabilities (
            boost::multi_array< T, R > inputMatrix )  [private]
```

The method getProbabilities calculates the probabilities for every element and stores the probabilities in the vector probElements

Parameters

in	*inputMatrix*	the original matrix of the VOI

The documentation for this class was generated from the following file:

- intensityHistogram.h

10.34 IntensityVolumeFeatures< T, R > Class Template Reference

Public Member Functions

- void getFractionalVolume (boost::multi_array< T, R > inputMatrix, vector< T > vectorMatrElem)
- void getGreyLevelFraction (boost::multi_array< T, R > inputMatrix)
- void **calculateAllIntensVolFeatures** (IntensityVolumeFeatures< T, R > &intVolFeatures, boost::multi_array< T, R > inputMatrix, vector< T > vectorMatrElem)
- void **writeCSVFileIntVol** (IntensityVolumeFeatures< T, R > intVol, string outputFolder)
- void **writeOneFileIntVol** (IntensityVolumeFeatures< T, R > intVol, string outputFolder)

Private Member Functions

- T getVolumeAtIntFraction (double percent)
- T getIntAtVolFraction (double percent, vector< T > diffGreyLevels)
- void **defineIntVolFeatures** (vector< string > &features)
- void **extractIntVolData** (vector< T > &intVolData, IntensityVolumeFeatures< T, R > intVolFeatures)

Private Attributes

- vector< T > **diffGreyLevels**
- T **maxGreyLevel**
- T **minGreyLevel**
- vector< T > **greyLevelFraction**
- vector< T > **fracVolume**
- T **volAtIntFrac10**
- T **volAtIntFrac90**
- T **intAtVolFrac10**
- T **intAtVolFrac90**
- T **diffVolAtIntFrac**
- T **diffIntAtVolFrac**

10.34.1 Member Function Documentation

10.34.1.1 getFractionalVolume()

```
template<class T , size_t R>
void IntensityVolumeFeatures< T, R >::getFractionalVolume (
            boost::multi_array< T, R > inputMatrix,
            vector< T > vectorMatrElem )
```

In the function getFractionalVolume the fractional volume of each grey level is calculated.
The vector fractional volume vector is filled in this function [in]: boost multi_array input matrix: matrix containing intensity values of VOI [in] vectorMatrElemen: vector containing all grey levels of VOI

10.34.1.2 getGreyLevelFraction()

```
template<class T , size_t R>
void IntensityVolumeFeatures< T, R >::getGreyLevelFraction (
            boost::multi_array< T, R > inputMatrix )
```

In the function getGreyLevelFraction the grey level fraction is calculated and appended to the vector greyLevel↩ Fraction.
[in]: boost multi_array input matrix: matrix containing intensity values of VOI [in] vectorMatrElemen: vector containing all grey levels of VOI

10.34.1.3 getIntAtVolFraction()

```
template<class T , size_t R>
T IntensityVolumeFeatures< T, R >::getIntAtVolFraction (
            double percent,
            vector< T > diffGreyLevels )  [private]
```

In the function getIntAtVolFraction calculates the intensity at a certain volume fraction for a certain percentage value. [in] double percent: percentage value for which the volume fraction is calculated

10.34.1.4 getVolumeAtIntFraction()

```
template<class T , size_t R>
T IntensityVolumeFeatures< T, R >::getVolumeAtIntFraction (
            double percent )  [private]
```

In the function getVolumeAtIntFraction calculates the volume at a certain intensity fraction for a certain percentage value. [in] double percent: percentage value for which the volume fraction is calculated

The documentation for this class was generated from the following file:

- intensityVolumeFeatures.h

10.35 LocalIntensityFeatures< T, R > Class Template Reference

```
#include <localIntensityFeatures.h>
```

Public Member Functions

- void **calculateAllLocalIntensityFeatures** (LocalIntensityFeatures< T, R > &localInt, Image< T, R > imageAttr, ConfigFile config)
- void **writeCSVFileLocalIntensity** (LocalIntensityFeatures< T, R > localInt, string outputfolder)
- void **writeOneFileLocalInt** (LocalIntensityFeatures< T, R > localInt, string outputfolder)

Private Types

- typedef boost::multi_array< T, R > **locMatrix**

Private Member Functions

- vector< vector< int > > getIndexOfMax (boost::multi_array< T, R > inputMatrix)
- void getConvMatrixSize (ImageType::Pointer mask, int &nrVoxelsDirection, float spacing, float radius)
- void fillConvMatrix (boost::multi_array< T, R > &matrix, ImageType::Pointer mask)
- void **fillVector** (vector< double > &index, boost::multi_array< T, R > convMatrix)
- ImageType::Pointer calculatePeaks (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > localIntMatrix, boost::multi_array< T, R > convolutionalMatrix, ImageType::Pointer image)
- boost::multi_array< T, R > calculateConvolutionMatrix (ImageType::Pointer mask)
- float **calculatePeakValues** (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > convolutionalMatrix, vector< int > nrValues, ImageType::Pointer image)
- void calculateLocalIntensityPeak (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > peak↩Matrix)
- void calculateGlobalIntensityPeak (boost::multi_array< T, R > peakMatrix, boost::multi_array< T, R > resegmentedMask)
- void **defineLocalIntenseFeatures** (vector< string > &features)
- void **extractLocalIntenseData** (vector< T > &localIntData, LocalIntensityFeatures< T, R > localInt↩Features)

Private Attributes

- T **maxElement**
- T **localIntensityPeak**
- T **globalIntensityPeak**
- float **spacingX**
- float **spacingY**
- float **spacingZ**
- float **volVoxel**
- MorphologicalFeatures< T, R > **morpho**
- float **pi** = 3.1415926535
- float **originalRadius** = 6.2
- float **voxelSize** [3]
- vector< T > **intValuesInCircle**

10.35.1 Detailed Description

template<class T, size_t R>
class LocalIntensityFeatures< T, R >

In the class LocalIntensityFeatures only the local and global peak are calculated.
In order to calculate global/local peak, a convolutional matrix is calculated. Multiplying by this matrix gives the circle around the present voxel.
Herefore, dependent on voxel size, the size of the convolutional matrix is calculated.
The calculation of the feature values is done before discretizing the matrix values to a user specified bin number.

10.35.2 Member Function Documentation

10.35.2.1 calculateConvolutionMatrix()

```
template<class T , size_t R>
boost::multi_array< T, R > LocalIntensityFeatures< T, R >::calculateConvolutionMatrix (
            ImageType::Pointer mask )  [private]
```

In the function calculateConvolutionMatrix, the convolutional matrix is filled using the function fillConvMatrix [in]: ImageType mask: image mask [out]: boost multi_array convolutional matrix

10.35.2.2 calculateGlobalIntensityPeak()

```
template<class T , size_t R>
void LocalIntensityFeatures< T, R >::calculateGlobalIntensityPeak (
            boost::multi_array< T, R > peakMatrix,
            boost::multi_array< T, R > resegmentedMask )  [private]
```

In the function calculateGlobalIntensityPeak the global intensity peak is calculated using the peak matrix.
[in]: boost multi_array peak matrix

10.35.2.3 calculateLocalIntensityPeak()

```
template<class T , size_t R>
void LocalIntensityFeatures< T, R >::calculateLocalIntensityPeak (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > peakMatrix )  [private]
```

In the function calculateLocalIntensityPeak the local intensity peak is calculated using the peak matrix.
[in]: boost multi_array input matrix [in]: boost multi_array peak matrix

10.35.2.4 calculatePeaks()

```
template<class T , size_t R>
ImageType::Pointer LocalIntensityFeatures< T, R >::calculatePeaks (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > localIntMatrix,
            boost::multi_array< T, R > convolutionalMatrix,
            ImageType::Pointer image )  [private]
```

In the function calculatePeaks calculates a matrix.
Every matrix element is the corresponding peak value to the element in the input matrix.
[in]: boost multi_array input matrix [in]: boost multi_array convolutional matrix [out]: boost multi_array peak matrix

10.35.2.5 fillConvMatrix()

```
template<class T , size_t R>
void LocalIntensityFeatures< T, R >::fillConvMatrix (
             boost::multi_array< T, R > & convMatrix,
             ImageType::Pointer mask )  [private]
```

In the function fillConvMatrix, we fill the convolutional matrix
For this, we start in the center of the matrix. In the center of the matrix, the radius is the actual radius of the sphere. Then we go one slice in x-direction. The radius changes, as we want to have a sphere with center in the center of the matrix.
The matrix is filled slice by slice.
[in]: boost multi_array convMatrix: the convolutional matrix is given as reference [in]: ImageType::Pointer mask: the image mask

10.35.2.6 getConvMatrixSize()

```
template<class T , size_t R>
void LocalIntensityFeatures< T, R >::getConvMatrixSize (
             ImageType::Pointer mask,
             int & nrVoxelsDirection,
             float spacing,
             float radius )  [private]
```

In the function getConvMatrixSize, it is determined which size the convolutional matrix will have [in]: ImageType::↩ Pointer mask: image mask [in]: int nrVoxelsDirection: as reference: how many vel we go in one direction [in]: float spacing: image spacing [in]: float radius: radius of circle

10.35.2.7 getIndexOfMax()

```
template<class T , size_t R>
vector< vector< int > > LocalIntensityFeatures< T, R >::getIndexOfMax (
             boost::multi_array< T, R > inputMatrix )  [private]
```

In the function getIndexOfMax all indices of the voxels that yield the maximum intensity value are stored in a vector. Herefore, a search is performed in the input matrix in order to look for the maximum element.
[in]: boost::multi_array inputMatrix [out]: vector containing indices

The documentation for this class was generated from the following file:

- localIntensityFeatures.h

10.36 MorphologicalFeatures< T, R > Class Template Reference

Public Member Functions

- void **calculateAllMorphologicalFeatures** (MorphologicalFeatures< T, R > &morphFeatures, Image< float, 3 > imageAttr, ConfigFile config)
- void **writeCSVFileMorphological** (MorphologicalFeatures< T, R > morph, string outputFolder)
- void **writeOneFileMorphological** (MorphologicalFeatures< T, R > morph, string outputFolder)

Public Attributes

- vector< T > **cartCoordinateROI**
- vector< T > **voxVolumeROI**
- const double **pi** = 3.141592653589793238463
- boost::multi_array< vector< T >, R > **coordinatesMatrix**

Private Types

- typedef itk::DefaultStaticMeshTraits< float, 3, 3, float, float, float > **MeshTrait**
- typedef itk::Mesh< float, 3, MeshTrait > **MeshType**
- typedef itk::BinaryMask3DMeshSource< ImageType, MeshType > **MeshSourceType**
- typedef unsigned char **PixelType**
- typedef MeshType::CellType **CellType**
- typedef MeshType::CellsContainer::ConstIterator **Celliterator**
- typedef itk::VTKPolyDataReader< MeshType > **ReaderTypeVTK**
- typedef ReaderTypeVTK::PointType **PointType**
- typedef itk::ConnectedComponentImageFilter< intImage, intImage > **ConnectedComponentFilterType**
- typedef itk::LabelImageToShapeLabelMapFilter< intImage > **LabelImageToShapeLabelMapFilterType**
- typedef int **LabelType**
- typedef itk::ShapeLabelObject< LabelType, R > **ShapeLabelObjectType**
- typedef itk::LabelMap< ShapeLabelObjectType > **LabelMapType**

Private Member Functions

- float calculateEuclideanDistance (int rowDiff, int colDiff, int depthDiff)
- vector< float > convertToCoordinates (int row, int col, int depth)
- ImageType::Pointer subsampleImage (ImageType::Pointer image, int factor)
- void getSurface (ImageType::Pointer mask)
- float calculateSurface (vector< vector< float > > vec)
- void getLabelObjectFeatures (ImageType::Pointer mask)
- void getBoundingBoxValues (ImageType::Pointer mask)
- void calculateVADensity (float &volDensity, float &areaDensity, itk::Size< R > regionSize)
- void calculateApproximateVolume (boost::multi_array< T, R > inputMatrix, vector< T > vectorOfMatr↩ Elements)
- void **calculateSurface2Volume** ()
- void **calculateCompactness1** ()
- void **calculateCompactness2** ()
- void **calculateSphericalDisproportion** ()
- void **calculateSphericity** ()
- void **calculateAsphericity** ()
- void **calculateMajorAxisLength** ()
- void **calculateMinorAxisLength** ()
- void **calculateLeastAxisLength** ()
- void **calculateElongation** ()
- void **calculateFlatness** ()
- void calculateMoransI (boost::multi_array< T, R > inputMatrix)
- void calculateIntegratedIntensity (vector< T > vectorOfMatrixElements)
- void calculateCentreOfMassShift (boost::multi_array< T, R > inputMatrix, vector< T > vectorOfMatr↩ Elements)
- int binomialCoefficient (int n, int k)
- float legendrePolynom (float x, int exponent)
- void calculateVolDensityAEE ()
- void calculateAreaDensityAEE ()
- void calculateVolDensityMEE ()
- void calculateAreaDensityMEE ()
- void **defineMorphologicalFeatures** (vector< string > &features)
- void **extractMorphologicalData** (vector< T > &morphData, MorphologicalFeatures< T, R > morph↩ Features)

Private Attributes

- ImageType::Pointer **image**
- ImageType::Pointer **mask**
- float **volume**
- float **appVolume**
- float **surface**
- float **surface2volumeRatio**
- float **compactness1**
- float **compactness2**
- float **sphericalDisproportion**
- float **sphericity**
- float **asphericity**
- float **majorAxisLength**
- float **minorAxisLength**
- float **leastAxisLength**
- float **maximumDiameter**
- float **flatness**
- float **elongation**
- T **integratredInt**
- T **centerOfMassShift**
- T **meanValue**
- T **absMean**
- float **moransI**
- float **gearysC**
- int **nrPixels**
- float **volDensityAEE**
- float **areaDensityAEE**
- float **volDensityOMBB**
- float **areaDensityOMBB**
- float **volDensityAABB**
- float **areaDensityAABB**
- float **volDensityMEE**
- float **areaDensityMEE**
- vector< vector< float > > **vec**
- vector< float > **coordinates**
- itk::Vector< double, 3 > **principalMoments**
- ImageType::PointType **origin**
- double **imageSpacingX**
- double **imageSpacingY**
- double **imageSpacingZ**

10.36.1 Member Function Documentation

10.36.1.1 binomialCoefficient()

```
template<class T , size_t R>
int MorphologicalFeatures< T, R >::binomialCoefficient (
            int n,
            int k )  [private]
```

In order to calculate the legendre polynom, the binomial coefficients are needed.

10.36.1.2 calculateApproximateVolume()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::calculateApproximateVolume (
            boost::multi_array< T, R > inputMatrix,
            vector< T > vectorOfMatrElement )  [private]
```

To calculate the volume, we count the voxels present in the image The amount of voxels is multiplied by the volume of one voxel

10.36.1.3 calculateAreaDensityAEE()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::calculateAreaDensityAEE ( )  [private]
```

The area density of the aligned enclosing ellipsoid is calculated in this function.
Herefore, the minor, major and least axis length are needed which have already been calculated before.
Furthermore the legendre polynom is used in this function. As it is enough to sum the first 20 parts of this polynom, we only calculate them.

10.36.1.4 calculateAreaDensityMEE()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::calculateAreaDensityMEE ( )  [private]
```

The area density of the minimal enclosing ellipsoid is calculated in this function.
Herefore, the minor, major and least axis length are needed which have already been calculated before.

10.36.1.5 calculateCentreOfMassShift()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::calculateCentreOfMassShift (
            boost::multi_array< T, R > inputMatrix,
            vector< T > vectorOfMatrElements )  [private]
```

In the function calculateCentreOfMassShift, the center of mass shift is calculated, taking as input the original matrix and a vector containing the matrix elements

10.36.1.6 calculateEuclideanDistance()

```
template<class T , size_t R>
float MorphologicalFeatures< T, R >::calculateEuclideanDistance (
            int rowDiff,
            int colDiff,
            int depthDiff )  [private]
```

In the function calculateEuclideanDistance the euclidean distance is calculated.
As input the function gets differences of row, columns and depth values and the euclidean distance is calculated using this values.

Parameters

in		

10.36.1.7 calculateIntegratedIntensity()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::calculateIntegratedIntensity (
            vector< T > vectorOfMatrixElements )  [private]
```

To calculate the integrated intensity, we calculate the mean value of the VOI and mulitplicate the value by the volume

10.36.1.8 calculateMoransI()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::calculateMoransI (
            boost::multi_array< T, R > inputMatrix )  [private]
```

In the function calculateMoransI, MoransI and Gearys C are calculated.
If the VOI is very big, the image is downsampled before this calculation to avoid very long computational time.
Still, for big VOIs, the calculation of these two features can take some time.

10.36.1.9 calculateSurface()

```
template<class T , size_t R>
float MorphologicalFeatures< T, R >::calculateSurface (
            vector< vector< float > > vec )  [private]
```

In the function calculateSurface the surface of the mask is calculated As input it requires a vector containing all coordinates of the mesh. From this coordinates it calculates the surface.

Parameters

in	*vector<vector<float>>*	vec: vector containing coordinates
out	*float*	surface

10.36.1.10 calculateVADensity()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::calculateVADensity (
            float & volDensity,
            float & areaDensity,
            itk::Size< R > regionSize )  [private]
```

In the function calculateVADensity the volume and the area density of a bounding box are calculated, given the size of the bounding box.

10.36.1.11 calculateVolDensityAEE()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::calculateVolDensityAEE ( )  [private]
```

Calculate the volume density of the aligned enclosing ellipsoid.

10.36.1.12 calculateVolDensityMEE()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::calculateVolDensityMEE ( )  [private]
```

The volume density of the minimal enclosing ellipsoid is calculated in this function.
Herefore, the minor, major and least axis length are needed which have already been calculated before.

10.36.1.13 convertToCoordinates()

```
template<class T , size_t R>
vector< float > MorphologicalFeatures< T, R >::convertToCoordinates (
            int row,
            int col,
            int depth )  [private]
```

In the function convertToCoordinates we convert the row, col and depth value of a voxel in a matrix to image coordinates

Parameters

in	*int*	row, int col, int depth: position of voxel in matrix
out	*vector<float>*	coordinates: image coordinates of these points

10.36.1.14 getBoundingBoxValues()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::getBoundingBoxValues (
            ImageType::Pointer mask )  [private]
```

In the function getBoundingBoxValues the bounding box region is extracted from the image
From this region, volume and surface are extracted in order to calculate volume and area density.

10.36.1.15 getLabelObjectFeatures()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::getLabelObjectFeatures (
            ImageType::Pointer mask )  [private]
```

In the function getLabelObjectFeatures a set of morphological features is calculated using the label image to shape label map filter

10.36.1.16 getSurface()

```
template<class T , size_t R>
void MorphologicalFeatures< T, R >::getSurface (
            ImageType::Pointer mask )  [private]
```

In the function getSurface the surface of the mask is calculated For this, the mask is converted to a mesh using the BinaryMask3DMeshSource filter From this mesh, the surface of every mesh cell is calculated

10.36.1.17 legendrePolynom()

```
template<class T , size_t R>
float MorphologicalFeatures< T, R >::legendrePolynom (
            float x,
            int exponent )  [private]
```

In order to calculate the area density of the minimal enclosing ellipsoid, the legendre polynom is needed.

10.36.1.18 subsampleImage()

```
template<class T , size_t R>
ImageType::Pointer MorphologicalFeatures< T, R >::subsampleImage (
            ImageType::Pointer imageR,
            int factor )  [private]
```

In order ot accelerate the calculations of MoransI and GearysC for big VOIs, the image has to be downsampled

Parameters

in	*ImageType::Pointer*	image
in	*int*	factor: factor for downsampling
out	*ImageType::Pointer*	downsampled image

The documentation for this class was generated from the following file:

- morphologicalFeatures.h

10.37 Neighbor2D< T, R > Class Template Reference

Public Attributes

- vector< T > **neighborVoxels2D**
- vector< vector< int > > **index2D**
- vector< T > **neighborVoxels3D**
- vector< vector< int > > **index3D**

The documentation for this class was generated from the following file:

- neighbor2D.h

10.38 NGLDMFeatures< T, R > Class Template Reference

`#include <NGLDMFeatures.h>`

Inheritance diagram for NGLDMFeatures< T, R >:

Collaboration diagram for NGLDMFeatures< T, R >:

Public Member Functions

- int **findIndex** (vector< T > array, int size, T target)
- void **writeCSVFileNGLDM** (NGLDMFeatures< T, R > ngldmFeat, string outputFolder)
- void **writeOneFileNGLDM** (NGLDMFeatures< T, R > ngldmFeat, string outputFolder)
- void **calculateAllNGLDMFeatures2D** (NGLDMFeatures< T, R > &ngldmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< T > vectorMatrElem, ConfigFile config)
- void calculateDependenceCountEnergy (boost::multi_array< double, 2 > probMatrix)

  calculateDependenceCountEnergy
- void **defineNGLDMFeatures** (vector< string > &features)

Public Attributes

- double **dependenceCountEnergy**

Private Member Functions

- void **extractNGLDMData** (vector< T > &ngldmData, NGLDMFeatures< T, R > ngldmFeatures)
- boost::multi_array< double, 2 > getMatrix (boost::multi_array< T, R > inputMatrix)

  getMatrix
- int getNeighborGreyLevels (boost::multi_array< T, R > inputMatrix, vector< int > actualIndex)

  getNeighborGreyLevels

Private Attributes

- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- vector< double > **rowSums**
- vector< double > **colSums**
- int **dist**
- int **coarseParam**

10.38.1 Detailed Description

template<class T, size_t R>
class NGLDMFeatures< T, R >

The class NGLDM is the class of the Neighborhood Grey Level Dependence Matrices.
A neighborhood are all voxels around one voxel within distance dist.
A voxel is dependent from the other, if $|X_c - X_m| < a$, where X_c is the center voxel and X_m are the other voxels in the neighborhood. The number of dependent voxels in a neighborhood is counted.
a is called the coarseness parameter.
The neighborhoods are checked for every voxel.
$s_=s(i,j)$ is the number of neighborhoods with center voxel with grey level i and dependece k = j-1
The most features are already defined in the GLRLM features.

10.38.2 Member Function Documentation

10.38.2.1 calculateDependenceCountEnergy()

```
template<class T , size_t R>
void NGLDMFeatures< T, R >::calculateDependenceCountEnergy (
            boost::multi_array< double, 2 > probMatrix )
```

calculateDependenceCountEnergy

Parameters

in	*boost*	multi array probMatrix: probability matrix filled

The function calculate the dependence count energy: $F_{countEnergy} = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g p_{ij}^2}$.

10.38.2.2 getMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > NGLDMFeatures< T, R >::getMatrix (
            boost::multi_array< T, R > inputMatrix )  [private]
```

getMatrix

Parameters

	boost	multi array inputMatrix: matrix filled with intensity values
out	*boost*	multi array: filled NGLD matrix

The function fills the NGLDMatrix with the corresponding values using the function getNeighborGreyLevels. It checks voxel by voxel the neighborhood in the distance that is set by the user.

10.38.2.3 getNeighborGreyLevels()

```
template<class T , size_t R>
int NGLDMFeatures< T, R >::getNeighborGreyLevels (
            boost::multi_array< T, R > inputMatrix,
            vector< int > actualIndex )  [private]
```

getNeighborGreyLevels

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*vector*	actualIndex : vector of the actual index
out	*ngldmnr*	: number of elements in neighborhood

The function checks the grey levels in the neighborhood and counts how many of them have the same intensity value.

The documentation for this class was generated from the following file:

- NGLDMFeatures.h

10.39 NGLDMFeatures2DAVG< T, R > Class Template Reference

`#include <NGLDMFeatures2DAVG.h>`

Inheritance diagram for NGLDMFeatures2DAVG< T, R >:

Collaboration diagram for NGLDMFeatures2DAVG< T, R >:

Public Member Functions

- void **writeCSVFileNGLDM2DAVG** (NGLDMFeatures2DAVG< T, R > ngldmFeat, string outputFolder)
- void **writeOneFileNGLDM2DAVG** (NGLDMFeatures2DAVG< T, R > ngldmFeat, string outputFolder)
- void **calculateAllNGLDMFeatures2DAVG** (NGLDMFeatures2DAVG< T, R > &ngldmFeatures, Image< T, R > imageAttr, boost::multi_array< T, R > ngldmMatrix, ConfigFile config)

Private Member Functions

- void **extractNGLDMData2DAVG** (vector< T > &ngldmData, NGLDMFeatures2DAVG< T, R > ngldm↩ Features)
- boost::multi_array< double, 2 > getMatrix (boost::multi_array< T, R > inputMatrix, boost::multi_array< T, R > ngldmNr, int depth)

  getMatrix
- int getNeighborGreyLevels (boost::multi_array< T, R > inputMatrix, vector< int > actualIndex)

  getNeighborGreyLevels

Private Attributes

- int **dist**
- int **coarseParam**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- vector< double > **rowSums**
- vector< double > **colSums**
- NGLDMFeatures2DMRG< T, R > **ngldm**

Additional Inherited Members

10.39.1 Detailed Description

template<class T, size_t R>
class NGLDMFeatures2DAVG< T, R >

The class NGLDM2DWOMerge inherites from the class NGLDMFeatures. Here the matrices are calculated slice by slice. For every slice the feature values are calculated.

10.39.2 Member Function Documentation

10.39.2.1 getMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > NGLDMFeatures2DAVG< T, R >::getMatrix (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > ngldmNr,
            int depth )  [private]
```

getMatrix

Parameters

	boost	multi array inputMatrix: matrix filled with intensity values
out	*boost*	multi array: filled NGLD matrix

The function fills the NGLDMatrix with the corresponding values using the function getNeighborGreyLevels. It checks voxel by voxel the neighborhood in the distance that is set by the user.

10.39.2.2 getNeighborGreyLevels()

```
template<class T , size_t R>
int NGLDMFeatures2DAVG< T, R >::getNeighborGreyLevels (
```

```
boost::multi_array< T, R > inputMatrix,
vector< int > actualIndex )  [private]
```

getNeighborGreyLevels

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*vector*	actualIndex : vector of the actual index
out	*ngldmnr*	: number of elements in neighborhood

The function checks the grey levels in the neighborhood and counts how many of them have the same intensity value.

The documentation for this class was generated from the following file:

- NGLDMFeatures2DAVG.h

10.40 NGLDMFeatures2DMRG< T, R > Class Template Reference

`#include <NGLDMFeatures2DMRG.h>`

Inheritance diagram for NGLDMFeatures2DMRG< T, R >:

Collaboration diagram for NGLDMFeatures2DMRG< T, R >:

Public Member Functions

- int **findIndex** (vector< T > array, int size, T target)
- void **writeCSVFileNGLDM2DMRG** (NGLDMFeatures2DMRG< T, R > ngldmFeat, string outputFolder)
- void **writeOneFileNGLDM2DMRG** (NGLDMFeatures2DMRG< T, R > ngldmFeat, string outputFolder)
- void **calculateAllNGLDMFeatures2DMRG** (NGLDMFeatures2DMRG< T, R > &ngldmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< T > vectorMatrElem, ConfigFile config)
- void calculateDependenceCountEnergy (boost::multi_array< double, 2 > probMatrix)

  calculateDependenceCountEnergy
- void **defineNGLDMFeatures** (vector< string > &features)

Public Attributes

- double **dependenceCountEnergy**

Private Member Functions

- void **extractNGLDMData** (vector< T > &ngldmData, NGLDMFeatures2DMRG< T, R > ngldmFeatures)
- boost::multi_array< double, 2 > getMatrix (boost::multi_array< T, R > inputMatrix)

  getMatrix
- int getNeighborGreyLevels (boost::multi_array< T, R > inputMatrix, vector< int > actualIndex)

  getNeighborGreyLevels

Private Attributes

- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- vector< double > **rowSums**
- vector< double > **colSums**
- int **dist**
- int **coarseParam**

10.40.1 Detailed Description

template<class T, size_t R>
class NGLDMFeatures2DMRG< T, R >

The class NGLDM is the class of the Neighborhood Grey Level Dependence Matrices.
A neighborhood are all voxels around one voxel within distance dist.
A voxel is dependent from the other, if $|X_c - X_m| < a$, where X_c is the center voxel and X_m are the other voxels in the neighborhood. The number of dependent voxels in a neighborhood is counted.
a is called the coarseness parameter.
The neighborhoods are checked for every voxel.
$s_{=}s(i,j)$ is the number of neighborhoods with center voxel with grey level i and dependece k = j-1
The most features are already defined in the GLRLM features.

10.40.2 Member Function Documentation

10.40.2.1 calculateDependenceCountEnergy()

```
template<class T , size_t R>
void NGLDMFeatures2DMRG< T, R >::calculateDependenceCountEnergy (
            boost::multi_array< double, 2 > probMatrix )
```

calculateDependenceCountEnergy

Parameters

in	*boost*	multi array probMatrix: probability matrix filled

The function calculate the dependence count energy: $F_{countEnergy} = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g p_{ij}^2}$.

10.40.2.2 getMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > NGLDMFeatures2DMRG< T, R >::getMatrix (
            boost::multi_array< T, R > inputMatrix )  [private]
```

getMatrix

Parameters

	boost	multi array inputMatrix: matrix filled with intensity values
out	*boost*	multi array: filled NGLD matrix

The function fills the NGLDMatrix with the corresponding values using the function getNeighborGreyLevels. It checks voxel by voxel the neighborhood in the distance that is set by the user.

10.40.2.3 getNeighborGreyLevels()

```
template<class T , size_t R>
int NGLDMFeatures2DMRG< T, R >::getNeighborGreyLevels (
            boost::multi_array< T, R > inputMatrix,
            vector< int > actualIndex )  [private]
```

getNeighborGreyLevels

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*vector*	actualIndex : vector of the actual index
out	*ngldmnr*	: number of elements in neighborhood

The function checks the grey levels in the neighborhood and counts how many of them have the same intensity value.

The documentation for this class was generated from the following file:

- NGLDMFeatures2DMRG.h

10.41 NGLDMFeatures2DWOMerge< T, R > Class Template Reference

`#include <NGLDMFeatures2DWOMerge.h>`

Inheritance diagram for NGLDMFeatures2DWOMerge< T, R >:

Collaboration diagram for NGLDMFeatures2DWOMerge< T, R >:

Public Member Functions

- void **writeCSVFileNGLDM** (NGLDMFeatures2DWOMerge< T, R > ngldmFeat, string outputFolder)
- void **writeOneFileNGLDM** (NGLDMFeatures2DWOMerge< T, R > ngldmFeat, string outputFolder)
- void **calculateAllNGLDMFeatures2DWOMerge2D** (NGLDMFeatures2DWOMerge< T, R > &ngldm↩ Features, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, ConfigFile config)

Private Member Functions

- void **extractNGLDMData2DWOMerge** (vector< T > &ngldmData, NGLDMFeatures2DWOMerge< T, R > ngldmFeatures)
- boost::multi_array< double, 2 > getMatrix (boost::multi_array< T, R > inputMatrix, int depth)

  getMatrix
- int getNeighborGreyLevels (boost::multi_array< T, R > inputMatrix, vector< int > actualIndex)

  getNeighborGreyLevels

Private Attributes

- int **dist**
- int **coarseParam**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- vector< double > **rowSums**
- vector< double > **colSums**
- NGLDMFeatures< T, R > **ngldm**

Additional Inherited Members

10.41.1 Detailed Description

template<class T, size_t R>
class NGLDMFeatures2DWOMerge< T, R >

The class NGLDM2DWOMerge inherites from the class NGLDMFeatures. Here the matrices are calculated slice by slice. For every slice the feature values are calculated.

10.41.2 Member Function Documentation

10.41.2.1 getMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > NGLDMFeatures2DWOMerge< T, R >::getMatrix (
            boost::multi_array< T, R > inputMatrix,
            int depth )  [private]
```

getMatrix

Parameters

	boost	multi array inputMatrix: matrix filled with intensity values
out	*boost*	multi array: filled NGLD matrix

The function fills the NGLDMatrix with the corresponding values using the function getNeighborGreyLevels. It checks voxel by voxel the neighborhood in the distance that is set by the user.

10.41.2.2 getNeighborGreyLevels()

```
template<class T , size_t R>
int NGLDMFeatures2DWOMerge< T, R >::getNeighborGreyLevels (
            boost::multi_array< T, R > inputMatrix,
            vector< int > actualIndex )  [private]
```

getNeighborGreyLevels

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*vector*	actualIndex : vector of the actual index
out	*ngldmnr*	: number of elements in neighborhood

The function checks the grey levels in the neighborhood and counts how many of them have the same intensity value.

The documentation for this class was generated from the following file:

- NGLDMFeatures2DWOMerge.h

10.42 NGLDMFeatures3D< T, R > Class Template Reference

`#include <NGLDMFeatures3D.h>`

Inheritance diagram for NGLDMFeatures3D< T, R >:

Collaboration diagram for NGLDMFeatures3D< T, R >:

Public Member Functions

- void **writeCSVFileNGLDM3D** (NGLDMFeatures3D< T, R > ngldmFeat, string outputFolder)
- void **writeOneFileNGLDM3D** (NGLDMFeatures3D< T, R > ngldmFeat, string outputFolder)
- void **calculateAllNGLDMFeatures3D** (NGLDMFeatures3D< T, R > &ngldmFeatures, boost::multi_array< T, R > inputMatrix, vector< T > diffGrey, vector< T > vectorMatrElem, ConfigFile config)

Private Member Functions

- void **defineNGLDMFeatures3D** (vector< string > &features)
- void **extractNGLDMData3D** (vector< T > &ngldmData, NGLDMFeatures3D< T, R > ngldmFeatures)
- boost::multi_array< double, 2 > getMatrix3D (boost::multi_array< T, R > inputMatrix)

  getMatrix3D
- int getNeighborGreyLevels3D (boost::multi_array< T, R > inputMatrix, vector< int > actualIndex)

  getNeighborGreyLevels3D

Private Attributes

- int **dist**
- int **coarseParam**
- vector< T > **sumProbRows**
- vector< T > **sumProbCols**
- vector< double > **rowSums**
- vector< double > **colSums**
- NGLDMFeatures2DMRG< T, R > **ngldm**

Additional Inherited Members

10.42.1 Detailed Description

template<class T, size_t R>
class NGLDMFeatures3D< T, R >

The class NGLDMFeatures3D inherites from the class NGLDMFeatures. The feature calculation are the same, only this matrix checks 3D neighborhoods.
A neighborhood are all voxels around one voxel within distance dist.
A voxel is dependent from the other, if $|X_c - X_m| < a$, where X_c is the center voxel and X_m are the other voxels in the neighborhood. The number of dependent voxels in a neighborhood is counted.
a is called the coarseness parameter.
The neighborhoods are checked for every voxel.
$s_=s(i,j)$ is the number of neighborhoods with center voxel with grey level i and dependece k = j-1
The most features are already defined in the GLRLM features.

10.42.2 Member Function Documentation

10.42.2.1 getMatrix3D()

```
template<class T , size_t R>
boost::multi_array< double, 2 > NGLDMFeatures3D< T, R >::getMatrix3D (
            boost::multi_array< T, R > inputMatrix )  [private]
```

getMatrix3D

Parameters

	boost	multi array inputMatrix: matrix filled with intensity values
out	*boost*	multi array: filled NGLD matrix

The function fills the NGLDMatrix with the corresponding values using the function getNeighborGreyLevels. It checks voxel by voxel the neighborhood in the distance that is set by the user.

10.42.2.2 getNeighborGreyLevels3D()

```
template<class T , size_t R>
int NGLDMFeatures3D< T, R >::getNeighborGreyLevels3D (
            boost::multi_array< T, R > inputMatrix,
            vector< int > actualIndex )  [private]
```

getNeighborGreyLevels3D

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*vector*	actualIndex : vector of the actual index
out	*ngldmnr*	: number of elements in neighborhood

The function checks the grey levels in the neighborhood and counts how many of them have the same intensity value.

The documentation for this class was generated from the following file:

- NGLDMFeatures3D.h

10.43 NGTDM2DAVG< T, R > Class Template Reference

`#include <NGTDM2DAVG.h>`

Inheritance diagram for NGTDM2DAVG< T, R >:

Collaboration diagram for NGTDM2DAVG< T, R >:

Public Member Functions

- void **getProbability** (vector< T > elementsOfWholeNeighborhood, boost::multi_array< double, 2 > &ngtd↩ Matrix)
- void **calculateAllNGTDMFeatures2DAVG** (NGTDM2DAVG< T, R > &ngtdmFeatures, Image< T, R > imageAttr, boost::multi_array< T, R > sumNeighborHoods, vector< double > spacing, ConfigFile config)
- void **writeCSVFileNGTDM2DAVG** (NGTDM2DAVG< T, R > ngtdm, string outputFolder)
- void **writeOneFileNGTDM2DAVG** (NGTDM2DAVG< T, R > ngtdm, string outputFolder)

Private Member Functions

- void **extractNGTDMData2DAVG** (vector< T > &ngtdmData, NGTDM2DAVG< T, R > ngtdmFeatures)
- boost::multi_array< double, 2 > getNGTDMatrix2DAVG (boost::multi_array< T, R > inputMatrix, boost↩ ::multi_array< T, R > sumNeighborHoods, int depth, ConfigFile config)

  getNGTDMatrix2DAVG In this function the NGTDM is filled.

Private Attributes

- NGTDMFeatures2DMRG< T, R > **ngtdm**
- vector< double > **actualSpacing**
- string **normNGTDM**
- int **dist**

10.43.1 Detailed Description

template<class T, size_t R>
class NGTDM2DAVG< T, R >

The class NGTDM inherites from the class NGTDM. The feature calculations are done according to the definitions in this class.
These matrices combine the sum of grey level differences of voxels with intensity value i and the average discretised grey levels of a neighborhood with distance dist from the actual voxel.
The average grey level within a neighborhood is defined as: $A_i = \frac{1}{W} \sum_{k_x=-dist}^{dist} \sum_{k_y=-dist}^{dist} \sum_{k_z=-dist}^{dist} X_{dgl}(j_x + k_x, j_y + k_y, j_z + k_z)$
where $k_x, k_y, k_z ! = 0$ and $W = (2dist + 1)$
Now, let n_i be the number of voxels with grey level i that have a complete neighborhood.
The entry in the NGTDM matrix is then: $s_i = \sum^{n_i} i - A_i, if n_i > 0$ and 0 otherwise.

10.43.2 Member Function Documentation

10.43.2.1 getNGTDMatrix2DAVG()

```
template<class T , size_t R>
boost::multi_array< double, 2 > NGTDM2DAVG< T, R >::getNGTDMatrix2DAVG (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > sumNeighborHoods,
            int depth,
            ConfigFile config )  [private]
```

getNGTDMatrix2DAVG In this function the NGTDM is filled.

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*int*	dist: size of neighborhood
in	*int*	depth: actual number of slice
out	*boost*	multi array: filled NGTD matrix The function fills the NGTDMatrix with the corresponding values

The documentation for this class was generated from the following file:

- NGTDM2DAVG.h

10.44 NGTDM2DWOMerge< T, R > Class Template Reference

`#include <NGTDM2DWOMerge.h>`

Inheritance diagram for NGTDM2DWOMerge< T, R >:

Collaboration diagram for NGTDM2DWOMerge< T, R >:

Public Member Functions

- void **getProbability** (vector< T > elementsOfWholeNeighborhood, boost::multi_array< double, 2 > &ngtd↩ Matrix)
- void **calculateAllNGTDMFeatures2DWOMerge** (NGTDM2DWOMerge< T, R > &ngtdm, boost::multi_↩ array< T, R > inputMatrix, vector< T > diffGrey, vector< double > spacing, ConfigFile config)
- void **writeCSVFileNGTDM2DWOMerge** (NGTDM2DWOMerge< T, R > ngtdm, string outputFolder)
- void **writeOneFileNGTDM2DWOMerge** (NGTDM2DWOMerge< T, R > ngtdm, string outputFolder)

Private Member Functions

- void **extractNGTDMData2DWOMerge** (vector< T > &ngtdmData, NGTDM2DWOMerge< T, R > ngtdm↩ Features)
- T getNeighborhood (boost::multi_array< T, R > inputMatrix, int *indexOfElement)

  getNeighborhood
- boost::multi_array< double, 2 > getNGTDMatrix2DWOMerge (boost::multi_array< T, R > inputMatrix, int depth)

  getNGTDMatrix2DWOMerge In this function the NGTDM is filled.

Private Attributes

- NGTDMFeatures< T, R > **ngtdm**
- vector< double > **actualSpacing**
- string **normNGTDM**
- int **dist**

10.44.1 Detailed Description

template<class T, size_t R>
class NGTDM2DWOMerge< T, R >

The class NGTDM inherites from the class NGTDM. The feature calculations are done according to the definitions in this class.
These matrices combine the sum of grey level differences of voxels with intensity value i and the average discretised grey levels of a neighborhood with distance dist from the actual voxel.
The average grey level within a neighborhood is defined as: $A_i = \frac{1}{W} \sum_{k_x=-dist}^{dist} \sum_{k_y=-dist}^{dist} \sum_{k_z=-dist}^{dist} X_{dgl}(j_x + k_x, j_y + k_y, j_z + k_z)$
where $k_x, k_y, k_z ! = 0$ and $W = (2dist + 1)$
Now, let n_i be the number of voxels with grey level i that have a complete neighborhood.
The entry in the NGTDM matrix is then: $s_i = \sum^{n_i} i - A_i, if n_i > 0$ and 0 otherwise.

10.44.2 Member Function Documentation

10.44.2.1 getNeighborhood()

```
template<class T , size_t R>
T NGTDM2DWOMerge< T, R >::getNeighborhood (
            boost::multi_array< T, R > inputMatrix,
            int * indexOfElement )  [private]
```

getNeighborhood

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*int*	indexOfElement: index of the actual element(for which the neighborhood is calculated)
in	*int*	dist: size of neighborhood
out	*T*	sum: average sume of all elements in the neighborhood except the center

The function get the values of all neighbors except the center of a certain element in a certain distance dist. It calculates the average sum of all these elements

10.44.2.2 getNGTDMatrix2DWOMerge()

```
template<class T , size_t R>
boost::multi_array< double, 2 > NGTDM2DWOMerge< T, R >::getNGTDMatrix2DWOMerge (
            boost::multi_array< T, R > inputMatrix,
            int depth )  [private]
```

getNGTDMatrix2DWOMerge In this function the NGTDM is filled.

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*int*	dist: size of neighborhood
in	*int*	depth: actual number of slice
out	*boost*	multi array: filled NGTD matrix The function fills the NGTDMatrix with the corresponding values

The documentation for this class was generated from the following file:

- NGTDM2DWOMerge.h

10.45 NGTDMFeatures< T, R > Class Template Reference

```
#include <NGTDM.h>
```

Inheritance diagram for NGTDMFeatures< T, R >:

Public Member Functions

- void **getProbability** (vector< T > elementsOfWholeNeighborhood, boost::multi_array< double, 2 > &ngtd↩ Matrix)
- double **calculateSumSi** (boost::multi_array< double, 2 > ngtdm)
- double **calculateSumSiPi** (boost::multi_array< double, 2 > ngtdm)
- int **getNGP** (boost::multi_array< double, 2 > ngtdm)
- int **getNV** (boost::multi_array< double, 2 > ngtdm)
- void **calculateStrength** (boost::multi_array< double, 2 > ngtdm)
- void **calculateComplexity** (boost::multi_array< double, 2 > ngtdm)
- void **calculateCoarseness** (boost::multi_array< double, 2 > ngtdm)
- void **calculateContrast** (boost::multi_array< double, 2 > ngtdm)
- void **calculateBusyness** (boost::multi_array< double, 2 > ngtdm)
- void **calculateAllNGTDMFeatures** (NGTDMFeatures< T, R > &ngtdm, boost::multi_array< T, R > input↩ Matrix, vector< T > diffGrey, vector< double > spacing, ConfigFile config)
- void **writeCSVFileNGTDM** (NGTDMFeatures< T, R > ngtdm, string outputFolder)
- void **writeOneFileNGTDM** (NGTDMFeatures< T, R > ngtdm, string outputFolder)
- void **defineNGTDMFeatures** (vector< string > &features)

Public Attributes

- vector< T > **diffGreyLevels**
- double **coarseness**
- double **contrast**
- double **busyness**
- double **complexity**
- double **strength**

Private Member Functions

- boost::multi_array< double, 2 > getNGTDMatrix (boost::multi_array< T, R > inputMatrix)
  getNGTDMatrix
- T getNeighborhood (boost::multi_array< T, R > inputMatrix, int *indexOfElement)
  getNeighborhood
- void **extractNGTDMData** (vector< T > &ngtdmData, NGTDMFeatures< T, R > ngtdmFeatures)

Private Attributes

- vector< double > **actualSpacing**
- string **normNGTDM**
- int **dist**

10.45.1 Detailed Description

template<class T, size_t R>
class NGTDMFeatures< T, R >

The class NGTDM is the class of the Neighborhood Grey Tone Difference Matrices.
These matrices combine the sum of grey level differences of voxels with intensity value i and the average discretised grey levels of a neighborhood with distance dist from the actual voxel.
The average grey level within a neighborhood is defined as: $A_i = \frac{1}{W} \sum_{k_x=-dist}^{dist} \sum_{k_y=-dist}^{dist} \sum_{k_z=-dist}^{dist} X_{dgl}(j_x + k_x, j_y + k_y, j_z + k_z)$
where $k_x, k_y, k_z ! = 0$ and $W = (2dist + 1)$
Now, let n_i be the number of voxels with grey level i that have a complete neighborhood.
The entry in the NGTDM matrix is then: $s_i = \sum^{n_i} i - A_i, if n_i > 0$ and 0 otherwise.

10.45.2 Member Function Documentation

10.45.2.1 getNeighborhood()

```
template<class T , size_t R>
T NGTDMFeatures< T, R >::getNeighborhood (
            boost::multi_array< T, R > inputMatrix,
            int * indexOfElement )  [private]
```

getNeighborhood

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*int*	indexOfElement: index of the actual element(for which the neighborhood is calculated)
in	*int*	dist: size of neighborhood
out	*T*	sum: average sume of all elements in the neighborhood except the center

The function get the values of all neighbors except the center of a certain element in a certain distance dist. It calculates the average sum of all these elements

10.45.2.2 getNGTDMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > NGTDMFeatures< T, R >::getNGTDMatrix (
            boost::multi_array< T, R > inputMatrix )  [private]
```

getNGTDMatrix

Parameters

	boost	multi array inputMatrix: matrix filled with intensity values
	int	dist: size of neighborhood
out	*boost*	multi array: filled NGTD matrix

The function fills the NGTDMatrix with the corresponding values

The documentation for this class was generated from the following file:

- NGTDM.h

10.46 NGTDMFeatures2DMRG< T, R > Class Template Reference

```
#include <NGTDM2DMRG.h>
```

Inheritance diagram for NGTDMFeatures2DMRG< T, R >:

Public Member Functions

- void **getProbability** (vector< T > elementsOfWholeNeighborhood, boost::multi_array< double, 2 > &ngtd↩ Matrix)
- double **calculateSumSi** (boost::multi_array< double, 2 > ngtdm)
- double **calculateSumSiPi** (boost::multi_array< double, 2 > ngtdm)
- int **getNGP** (boost::multi_array< double, 2 > ngtdm)
- int **getNV** (boost::multi_array< double, 2 > ngtdm)
- void **calculateStrength** (boost::multi_array< double, 2 > ngtdm)
- void **calculateComplexity** (boost::multi_array< double, 2 > ngtdm)
- void **calculateCoarseness** (boost::multi_array< double, 2 > ngtdm)
- void **calculateContrast** (boost::multi_array< double, 2 > ngtdm)
- void **calculateBusyness** (boost::multi_array< double, 2 > ngtdm)
- void **calculateAllNGTDMFeatures2DMRG** (NGTDMFeatures2DMRG< T, R > &ngtdm, Image< T, R > imageAttr, boost::multi_array< T, R > neighborHoodSum, vector< double > spacing, ConfigFile config)
- void **writeCSVFileNGTDM** (NGTDMFeatures2DMRG< T, R > ngtdm, string outputFolder)
- void **writeOneFileNGTDM** (NGTDMFeatures2DMRG< T, R > ngtdm, string outputFolder)
- void **defineNGTDMFeatures2DMRG** (vector< string > &features)

Public Attributes

- vector< T > **diffGreyLevels**
- double **coarseness**
- double **contrast**
- double **busyness**
- double **complexity**
- double **strength**

Private Member Functions

- boost::multi_array< double, 2 > getNGTDMatrix (boost::multi_array< T, R > inputMatrix, boost::multi_↩ array< T, R > neighborHoodSum)

  getNGTDMatrix
- void **extractNGTDMData** (vector< T > &ngtdmData, NGTDMFeatures2DMRG< T, R > NGTDM↩ Features2DMRG)

Private Attributes

- vector< double > **actualSpacing**
- string **normNGTDM**
- int **dist**

10.46.1 Detailed Description

template<class T, size_t R>
class NGTDMFeatures2DMRG< T, R >

The class NGTDM is the class of the Neighborhood Grey Tone Difference Matrices.
These matrices combine the sum of grey level differences of voxels with intensity value i and the average discretised grey levels of a neighborhood with distance dist from the actual voxel.
The average grey level within a neighborhood is defined as: $A_i = \frac{1}{W} \sum_{k_x=-dist}^{dist} \sum_{k_y=-dist}^{dist} \sum_{k_z=-dist}^{dist} X_{dgl}(j_x + k_x, j_y + k_y, j_z + k_z)$
where $k_x, k_y, k_z ! = 0$ and $W = (2dist + 1)$
Now, let n_i be the number of voxels with grey level i that have a complete neighborhood.
The entry in the NGTDM matrix is then: $s_i = \sum^{n_i} i - A_i, if n_i > 0$ and 0 otherwise.

10.46.2 Member Function Documentation

10.46.2.1 getNGTDMatrix()

```
template<class T , size_t R>
boost::multi_array< double, 2 > NGTDMFeatures2DMRG< T, R >::getNGTDMatrix (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > neighborHoodSum )  [private]
```

getNGTDMatrix

Parameters

	boost	multi array inputMatrix: matrix filled with intensity values
	int	dist: size of neighborhood
out	*boost*	multi array: filled NGTD matrix

The function fills the NGTDMatrix with the corresponding values

The documentation for this class was generated from the following file:

- NGTDM2DMRG.h

10.47 NGTDMFeatures3D< T, R > Class Template Reference

```
#include <NGTDM3D.h>
```

Inheritance diagram for NGTDMFeatures3D< T, R >:

Collaboration diagram for NGTDMFeatures3D< T, R >:

Public Member Functions

- void **getProbability** (vector< T > elementsOfWholeNeighborhood, boost::multi_array< double, 2 > &ngtd↩ Matrix)
- void **calculateAllNGTDMFeatures3D** (NGTDMFeatures3D< T, R > &ngtdm, Image< T, R > imageAttr, boost::multi_array< T, R > neighborMatrix, vector< double > spacing, ConfigFile config)
- void **writeCSVFileNGTDM3D** (NGTDMFeatures3D< T, R > ngtdmFeatures, string outputFolder)
- void **writeOneFileNGTDM3D** (NGTDMFeatures3D< T, R > ngtdmFeatures, string outputFolder)

Private Member Functions

- void **extractNGTDMData3D** (vector< T > &ngtdmData, NGTDMFeatures3D< T, R > ngtdmFeatures)
- T getNeighborhood3D (boost::multi_array< T, R > inputMatrix, int *indexOfElement)

  getNeighborhood3D
- boost::multi_array< double, 2 > getNGTDMatrix3D (boost::multi_array< T, R > inputMatrix, boost::multi_↩ array< T, R > neighborMatrix)

  getNGTDMatrix3D In this function the NGTDM is filled for the 3D case.

Private Attributes

- NGTDMFeatures2DMRG< T, R > **ngtdm**
- vector< double > **actualSpacing**
- string **normNGTDM**
- int **dist**

10.47.1 Detailed Description

template<class T, size_t R = 3>
class NGTDMFeatures3D< T, R >

The class NGTDM3D inherites from the class NGTDM. Here the matrix is calculated looking at the 3D VOI and not slice by slice.
All other definitions are the same.
These matrices combine the sum of grey level differences of voxels with intensity value i and the average discretised grey levels of a neighborhood with distance dist from the actual voxel.
The average grey level within a neighborhood is defined as: $A_i = \frac{1}{W} \sum_{k_x=-dist}^{dist} \sum_{k_y=-dist}^{dist} \sum_{k_z=-dist}^{dist} X_{dgl}(j_x + k_x, j_y + k_y, j_z + k_z)$
where $k_x, k_y, k_z ! = 0$ and $W = (2dist + 1)$
Now, let n_i be the number of voxels with grey level i that have a complete neighborhood.
The entry in the NGTDM matrix is then: $s_i = \sum^{n_i} i - A_i, if n_i > 0$ and 0 otherwise.

10.47.2 Member Function Documentation

10.47.2.1 getNeighborhood3D()

```
template<class T , size_t R>
T NGTDMFeatures3D< T, R >::getNeighborhood3D (
            boost::multi_array< T, R > inputMatrix,
            int * indexOfElement )  [private]
```

getNeighborhood3D

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*int*	indexOfElement: index of the actual element(for which the neighborhood is calculated)
in	*int*	dist: size of neighborhood
out	*T*	sum: average sume of all elements in the neighborhood except the center

The function get the values of all neighbors except the center of a certain element in a certain distance dist. It calculates the average sum of all these elements.
The function is analog to the getNeighborhood functions of the 2D case.

10.47.2.2 getNGTDMatrix3D()

```
template<class T , size_t R>
boost::multi_array< double, 2 > NGTDMFeatures3D< T, R >::getNGTDMatrix3D (
            boost::multi_array< T, R > inputMatrix,
            boost::multi_array< T, R > neighborMatrix )  [private]
```

getNGTDMatrix3D In this function the NGTDM is filled for the 3D case.

Parameters

in	*boost*	multi array inputMatrix: matrix filled with intensity values
in	*int*	dist: size of neighborhood
in	*int*	depth: actual number of slice
out	*boost*	multi array: filled NGTD matrix

The documentation for this class was generated from the following file:

- NGTDM3D.h

10.48 square_accumulate< T > Class Template Reference

Public Member Functions

- const T & **result** (void) const
- void **operator()** (const T &val)

Private Attributes

- T **_sum**

The documentation for this class was generated from the following file:

- helpFunctions.h

10.49 StatisticalFeatures< T, R > Class Template Reference

```
#include <statisticalFeatures.h>
```

Inheritance diagram for StatisticalFeatures< T, R >:

Public Member Functions

- void calculateMean (vector< T > vectorMatrElem)

  calculateMean
- void calculateVariance ()

  calculateVariance The variance value is calculated $F_{var} = \frac{1}{N_V} \sum (X_{gl} - \mu)^2$
- void calculateSkewness ()

  calculateSkewness The skewness of the intensity distribution is calculated $F_{skew} = \frac{\frac{1}{N_V} \sum (X_{gl} - \mu)^3}{(\frac{1}{N_V} \sum (X_{gl} - \mu)^2)^{\frac{3}{2}}}$
- void calculateKurtosis ()

  calculateKurtosis The kurtosis of the intensity distribution is calculated $F_{kur} = \frac{\frac{1}{N_V} \sum (X_{gl} - \mu)^4}{(\frac{1}{N_V} \sum (X_{gl} - \mu)^2)^2} - 3$
- void getMedian (std::vector< T > vectorMatrElement)

  getMedian The median of the intensity distribution is calculated
- void getMinimum (vector< T > matrixVector)

  getMinimum The smallest element of all intensity values in the VOI is extracted as F_{min}
- void getMaximum (vector< T > matrixVector)

  getMaximum The highest element of all intensity values in the VOI is extracted as F_{max}
- void getRange ()

  getRange The range of the distribution is defined as: $F_{range} = F_{max} - F_{min}$
- double getPercentile (vector< T > matrixVector, T probability)

  getPercentile
- void get10percentile (vector< T > matrixVector)

  get10Percentile
- void get90percentile (vector< T > matrixVector)

  get90Percentile
- void getInterquartileRange (vector< T > matrixVector)

  getInterquartileRange The interquartile range is defined as follows: $F_{intquarRange} = P_{75} - P_{25}$ *With* P_{75} *and* P_{25} *being the* 75^{th} *and* 25^{th} *percentile.*
- void getQuartileCoeff ()

  getQuartileCoeff The quartile coefficient of dispersion is defined as follows: $F_{quartCoeff} = \frac{P_{75} - P_{25}}{P_{75} + P_{25}}$ *With* P_{75} *and* P_{25} *being the* 75^{th} *and* 25^{th} *percentile.*
  It is another measurement for the dispersion of the distribution
- void getCoeffOfVar ()

  getCoeffOfVar The coefficient of variation is defined as follows: $F_{coeffOfVar} = \frac{\sigma}{\mu}$ *It measures for the dispersion of the distribution*
- void rootMeanSquare (vector< T > vectorMatrElem)

  rootMeanSquare The root mean square is defined as follows: $F_{rootMeanSquare} = \sqrt{\frac{\sum X_{gl}^2}{N_V}}$
- void energy (vector< T > vectorMatrElem)

  energy The energy of the distribution is defined as follows: $F_{energy} = \sum X_{gl}^2$
- void meanAbsoluteDev (vector< T > vectorMatrElem)

  meanAbsoulteDev The mean absolute deviation is calculated as: $F_{meanAbsoulteDev} = \frac{1}{N_V} \sum X_{gl,j}^2 - \mu$
- void medianAbsoluteDev (vector< T > vectorMatrElem)

  medianAbsoluteDev The median absolute deviation is defined as: $F_{medianAbsoulteDev} = \frac{1}{N_V} \sum X_{gl,j}^2 - F_{median}$
- void getRobustMeanAbsDev (vector< T > vectorMatrElem)

  getRobustMeanAbsDev Because outliers can have a big influence on the mean absolute deviation, the set of intensity values included in the calculation of the robust mean can be limited to: $X_{10-90} = \{x \in X_{gl} | P_{10}(X_{gl}) \leq x \leq P_{90}(X_{gl})\}$ *So, the set of grey levels is limited to the grey levels closer to the median of the distribution and the influence of outliers is minimized.*
  X_{10-90} *is the set of* N_{10-90} *grey level elements which lie in between (or are equal to) the* 10^{th} *and* 90^{th} *percentile.*
  The robust mean absolute deviation is calculated as follows: $F_{robmeanAbsDev} = \frac{1}{N_{10-90}} \sum X_{gl10-90,j} - X_{gl10-90,j}$
- void getGreaterElements (vector< T > &vectorOfMatrixElem, T &value)

getBiggerElements

- void getSmallerElements (vector< T > &vectorOfMatrixElem, T &value)

  getSmallerElements
- void **calculateAllStatFeatures** (StatisticalFeatures< T, R > &stat, vector< T > vectorMatrElement)
- void **writeCSVFileStatistic** (StatisticalFeatures< T, R > stat, string outputFolder)
- void **writeOneFileStatistic** (StatisticalFeatures< T, R > stat, string outputFolder)

Public Attributes

- T **meanValue**
- T **varianceValue**
- T **skewnessValue**
- T **kurtosisValue**
- T **medianValue**
- T **minimumValue**
- T **maximumValue**
- T **rangeValue**
- T **percentile10**
- T **percentile90**
- T **percentile25**
- T **percentile75**
- T **interquartileRange**
- T **quartileCoeff**
- T **coeffOfVar**
- T **energyValue**
- T **rootMean**
- T **meanAbsDev**
- T **medianAbsDev**
- T **robustMeanAbsDev**

Private Types

- typedef boost::accumulators::features< tag::mean, tag::variance, tag::median, tag::skewness, tag::kurtosis > **Features**
- typedef accumulator_set< T, Features > **Accumulator**

Private Member Functions

- void fillAccumulator (Accumulator &acc, vector< T > vectorMatrElem)

  fillAccumulator
- void **defineStatFeatures** (vector< string > &features)
- void **extractStatData** (vector< T > &statData, StatisticalFeatures< T, R > statFeatures)

Private Attributes

- vector< T > **vectorOfMatrixElem**
- Accumulator **acc**

10.49.1 Detailed Description

template<class T, size_t R>
class StatisticalFeatures< T, R >

Statistical Features describe the distribution of grey levels within the Volume of interest (VOI). This features are calculated without previous discretization.
The class Statistical Features calculates the statistical features of the whole 3D VOI.
Every features is an attribute of the class Statistical Features.
The feature values are calculated using the accumulator function of the boost library.

10.49.2 Member Function Documentation

10.49.2.1 calculateMean()

```
template<class T, size_t R>
void StatisticalFeatures< T, R >::calculateMean (
            vector< T > vectorMatrElem )
```

calculateMean

Parameters

in	*vectorOfMatrElement*	array containing all intensitiy values within the VOI The accumulator is filled calling the fillAccumulator function and the mean intensity value of the VOI is calculated $F_{mean} = \frac{1}{N_v} \sum X_{gl}$

10.49.2.2 fillAccumulator()

```
template<class T, size_t R>
void StatisticalFeatures< T, R >::fillAccumulator (
            Accumulator & acc,
            vector< T > vectorMatrElem )  [private]
```

fillAccumulator

Parameters

input	inputMatrix: matrix containing intensity values
input	Accumulator with different tags The intensity values of the image are copied in the accumulator.

10.49.2.3 get10percentile()

```
template<class T , size_t R>
void StatisticalFeatures< T, R >::get10percentile (
            vector< T > matrixVector )
```

get10Percentile

Parameters

in	*matrixVector*	vector containing all intensity values of the VOI The function calculates the 10^{th} percentile P_{10}.

10.49.2.4 get90percentile()

```
template<class T , size_t R>
void StatisticalFeatures< T, R >::get90percentile (
            vector< T > matrixVector )
```

get90Percentile

Parameters

in	*matrixVector*	vector containing all intensity values of the VOI The function calculates the 90^{th} percentile P_{90}.

10.49.2.5 getGreaterElements()

```
template<class T , size_t R>
void StatisticalFeatures< T, R >::getGreaterElements (
            vector< T > & vectorOfElem,
            T & value )
```

getBiggerElements

Parameters

in	*vectorOfElem*	vector of all elements in the VOI
in	*valueLimit*	limit, in the end all values higher than this value are stored in the vector

10.49.2.6 getPercentile()

```
template<class T , size_t R>
```

```
double StatisticalFeatures< T, R >::getPercentile (
            vector< T > matrixVector,
            T probability )
```

getPercentile

Parameters

in	*matrixVector*	vector containing all intensity values of the VOI
in	*probability*	probability of the percentile, that should be calculated The function is a help function to calculate percentiles of a certain probability It uses the accumulator function of the boost library together with the p_square_quantile-tag

10.49.2.7 getSmallerElements()

```
template<class T , size_t R>
void StatisticalFeatures< T, R >::getSmallerElements (
            vector< T > & vectorOfElem,
            T & valueLimit )
```

getSmallerElements

Parameters

in	*vectorOfElem*	vector of all elements in the VOI
in	*valueLimit*	limit, in the end all values smaller than this value are stored in the vector

The documentation for this class was generated from the following file:

- statisticalFeatures.h

10.50 sum_absol_value< T > Class Template Reference

Public Member Functions

- const T & **result** (void) const
- void **operator()** (const T &val)

Private Attributes

- T **_sum**

The documentation for this class was generated from the following file:

- helpFunctions.h

10.51 sum_robust< T > Class Template Reference

Public Member Functions

- const T & **result** (void) const
- void **operator()** (const T &val)

Private Attributes

- T **_sum**

The documentation for this class was generated from the following file:

- helpFunctions.h

Chapter 11

File Documentation

11.1 featureCalculation.h File Reference

```
#include "readDicom.h"
#include "readPrj.h"
#include "image.h"
#include "softwareParameters.h"
#include "featureCalculation.cpp"
```

Include dependency graph for featureCalculation.h: This graph shows which files directly or indirectly include this file:

Functions

- void prepareDataForFeatureCalculation (ConfigFile config)
- void **calculateFeaturesForConfig** (ImageType ∗imageFiltered, ImageType ∗maskNewSpacing, ConfigFile config)
- ImageType::Pointer **readVoiFilePET** (string prjPath, string voiPath, ImageType ∗image, ConfigFile config)
- void **writeImageData2Log** (ConfigFile config)

11.1.1 Function Documentation

11.1.1.1 prepareDataForFeatureCalculation()

```
void prepareDataForFeatureCalculation (
            ConfigFile config )
```

In the function prepareDataForFeatureCalculation, first the image and the mask are read. For this, the ITK-library is used.
After reading the mask, a bounding box from the region of interest is created.
The region of this bounding box is extracted from the image and the mask, which leads to smaller subimages. From these subimages, image attributes are extracted.

11.2 GLCMFeatures.h File Reference

```
#include <iostream>
#include "boost/multi_array.hpp"
#include "math.h"
#include <boost/range/algorithm.hpp>
#include "matrixFunctions.h"
#include "helpFunctions.h"
#include "image.h"
```

Include dependency graph for GLCMFeatures.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLCMFeatures< T, R >

11.3 GLCMFeatures2DAVG.h File Reference

```
#include "GLCMFeatures.h"
```

Include dependency graph for GLCMFeatures2DAVG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLCMFeatures2DAVG< T, R >

11.4 GLCMFeatures2DFullMerge.h File Reference

```
#include "GLCMFeatures.h"
```

Include dependency graph for GLCMFeatures2DFullMerge.h:

Classes

- class GLCMFeatures2DFullMerge< T, R >

11.5 GLCMFeatures2DMRG.h File Reference

```
#include "GLCMFeatures.h"
```

Include dependency graph for GLCMFeatures2DMRG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLCMFeatures2DMRG< T, R >

11.6 GLCMFeatures2DVMRG.h File Reference

`#include "GLCMFeatures.h"`
Include dependency graph for GLCMFeatures2DVMRG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLCMFeatures2DVMRG< T, R >

11.7 GLCMFeatures2DWMerge.h File Reference

`#include "GLCMFeatures.h"`
Include dependency graph for GLCMFeatures2DWMerge.h:

Classes

- class GLCMFeatures2DWMerge< T, R >

11.8 GLCMFeatures2DWOMerge.h File Reference

`#include "GLCMFeatures.h"`
Include dependency graph for GLCMFeatures2DWOMerge.h:

Classes

- class GLCMFeatures2DWOMerge< T, R >

11.9 GLCMFeatures3D.h File Reference

`#include "GLCMFeatures.h"`
Include dependency graph for GLCMFeatures3D.h:

Classes

- class GLCMFeatures3D< T, R >

11.10 GLCMFeatures3DAVG.h File Reference

`#include "GLCMFeatures3DMRG.h"`
Include dependency graph for GLCMFeatures3DAVG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLCMFeatures3DAVG< T, R >

11.11 GLCMFeatures3DMRG.h File Reference

```
#include "GLCMFeatures.h"
```
Include dependency graph for GLCMFeatures3DMRG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLCMFeatures3DMRG< T, R >

11.12 GLCMFeatures3DWMerge.h File Reference

```
#include "GLCMFeatures.h"
```
Include dependency graph for GLCMFeatures3DWMerge.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLCMFeatures3DWMerge< T, R >

11.13 GLCMFeatures3DWOMerge.h File Reference

```
#include "GLCMFeatures3DWMerge.h"
```
Include dependency graph for GLCMFeatures3DWOMerge.h:

Classes

- class GLCMFeatures3DWOMerge< T, R >

11.14 GLDZMFeatures2D.h File Reference

```
#include <algorithm>
#include "GLSZMFeatures2D.h"
```
Include dependency graph for GLDZMFeatures2D.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLDZMFeatures2D< T, R >

11.15 GLDZMFeatures2DAVG.h File Reference

```
#include "GLDZMFeatures3D.h"
#include "itkNeighborhoodIterator.h"
#include "itkNeighborhoodOperatorImageFunction.h"
```

Include dependency graph for GLDZMFeatures2DAVG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLDZMFeatures2DAVG< T, R >

11.16 GLDZMFeatures2DMRG.h File Reference

```
#include <algorithm>
#include "GLSZMFeatures2D.h"
```

Include dependency graph for GLDZMFeatures2DMRG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLDZMFeatures2D< T, R >

11.17 GLDZMFeatures2DWOMerge.h File Reference

```
#include "GLDZMFeatures2D.h"
```

Include dependency graph for GLDZMFeatures2DWOMerge.h:

Classes

- class GLDZMFeatures2DWOMerge< T, R >

11.18 GLDZMFeatures3D.h File Reference

```
#include "GLDZMFeatures2DMRG.h"
```

Include dependency graph for GLDZMFeatures3D.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLDZMFeatures3D< T, R >

11.19 GLRLMFeatures.h File Reference

```
#include "matrixFunctions.h"
#include "image.h"
#include "helpFunctions.h"
#include <iostream>
#include <algorithm>
#include <vector>
```

Include dependency graph for GLRLMFeatures.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLRLMFeatures< T, R >

11.20 GLRLMFeatures2DAVG.h File Reference

```
#include "GLRLMFeatures2DVMRG.h"
```

Include dependency graph for GLRLMFeatures2DAVG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLRLMFeatures2DAVG< T, R >

11.21 GLRLMFeatures2DFullMerge.h File Reference

```
#include "GLRLMFeatures.h"
```

Include dependency graph for GLRLMFeatures2DFullMerge.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLRLMFeatures2DFullMerge< T, R >

11.22 GLRLMFeatures2DMRG.h File Reference

```
#include "GLRLMFeatures.h"
```

Include dependency graph for GLRLMFeatures2DMRG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLRLMFeatures2DMRG< T, R >

11.23 GLRLMFeatures2DVMRG.h File Reference

`#include "GLRLMFeatures.h"`
Include dependency graph for GLRLMFeatures2DVMRG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLRLMFeatures2DVMRG< T, R >

11.24 GLRLMFeatures2DWMerge.h File Reference

`#include "GLRLMFeatures.h"`
Include dependency graph for GLRLMFeatures2DWMerge.h:

Classes

- class GLRLMFeatures2DWMerge< T, R >

11.25 GLRLMFeatures2DWOMerge.h File Reference

`#include "GLRLMFeatures2DFullMerge.h"`
Include dependency graph for GLRLMFeatures2DWOMerge.h:

Classes

- class GLRLMFeatures2DWOMerge< T, R >

11.26 GLRLMFeatures3D.h File Reference

`#include "GLRLMFeatures3DAVG.h"`
Include dependency graph for GLRLMFeatures3D.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLRLMFeatures3D< T, R >

11.27 GLRLMFeatures3DAVG.h File Reference

`#include "GLRLMFeatures.h"`
Include dependency graph for GLRLMFeatures3DAVG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLRLMFeatures3DAVG< T, R >

11.28 GLRLMFeatures3DWOMerge.h File Reference

`#include "GLRLMFeatures.h"`
Include dependency graph for GLRLMFeatures3DWOMerge.h:

Classes

- class GLRLMFeatures3DWOMerge< T, R >

11.29 GLSZMFeatures2D.h File Reference

`#include "GLRLMFeatures.h"`
Include dependency graph for GLSZMFeatures2D.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLSZMFeatures2DMRG< T, R >

11.30 GLSZMFeatures2DAVG.h File Reference

`#include "GLSZMFeatures2D.h"`
Include dependency graph for GLSZMFeatures2DAVG.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLSZMFeatures2DAVG< T, R >

11.31 GLSZMFeatures2DWOMerge.h File Reference

`#include "GLSZMFeatures2D.h"`
Include dependency graph for GLSZMFeatures2DWOMerge.h:

Classes

- class GLSZMFeatures2DWOMerge< T, R >

11.32 GLSZMFeatures3D.h File Reference

```
#include "GLSZMFeatures2D.h"
```

Include dependency graph for GLSZMFeatures3D.h: This graph shows which files directly or indirectly include this file:

Classes

- class GLSZMFeatures3D< T, R >

11.33 image.h File Reference

```
#include <iostream>
#include "math.h"
#include <string>
#include <algorithm>
#include <vector>
#include "boost/multi_array.hpp"
#include "itkMesh.h"
#include "itkBinaryMask3DMeshSource.h"
#include "itkBinaryThresholdImageFilter.h"
#include "itkSimplexMesh.h"
#include "itkSimplexMeshVolumeCalculator.h"
#include "itkTriangleMeshToSimplexMeshFilter.h"
#include "itkLabelObject.h"
#include "itkLabelMap.h"
#include "itkLabelImageToLabelMapFilter.h"
#include "itkLabelMapToLabelImageFilter.h"
#include "itkTypes.h"
#include <itkImageFileWriter.h>
#include "readConfigFile.h"
```

Include dependency graph for image.h: This graph shows which files directly or indirectly include this file:

Classes

- class Image< T, R >

11.34 intensityHistogram.h File Reference

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <cmath>
#include <boost/accumulators/accumulators.hpp>
#include <boost/accumulators/statistics/density.hpp>
#include <boost/accumulators/statistics/stats.hpp>
#include "statisticalFeatures.h"
```

Include dependency graph for intensityHistogram.h: This graph shows which files directly or indirectly include this file:

Classes

- class IntensityHistogram< T, R >

11.35 intensityVolumeFeatures.h File Reference

```
#include <iostream>
#include <vector>
#include "boost/multi_array.hpp"
#include "boost/range/combine.hpp"
#include "boost/foreach.hpp"
#include "image.h"
```

Include dependency graph for intensityVolumeFeatures.h: This graph shows which files directly or indirectly include this file:

Classes

- class IntensityVolumeFeatures< T, R >

11.36 localIntensityFeatures.h File Reference

```
#include "itkConvolutionImageFilter.h"
#include "image.h"
#include "morphologicalFeatures.h"
#include "matrixFunctions.h"
```

Include dependency graph for localIntensityFeatures.h: This graph shows which files directly or indirectly include this file:

Classes

- class LocalIntensityFeatures< T, R >

11.37 morphologicalFeatures.h File Reference

```
#include <cmath>
#include "matrixFunctions.h"
#include <itkImage.h>
#include <itkImageFileReader.h>
#include "itkVTKPolyDataReader.h"
#include "itkMesh.h"
#include "itkBinaryMask3DMeshSource.h"
#include "itkBinaryThresholdImageFilter.h"
#include "itkSimplexMesh.h"
#include "itkSimplexMeshVolumeCalculator.h"
#include "itkTriangleMeshToSimplexMeshFilter.h"
#include "itkLabelImageToShapeLabelMapFilter.h"
#include "itkConnectedComponentImageFilter.h"
#include "itkImageMaskSpatialObject.h"
#include "itkCastImageFilter.h"
#include "image.h"
#include "itkTypes.h"
```

Include dependency graph for morphologicalFeatures.h: This graph shows which files directly or indirectly include this file:

Classes

- class MorphologicalFeatures< T, R >

11.38 NGLDMFeatures.h File Reference

`#include "GLRLMFeatures.h"`
Include dependency graph for NGLDMFeatures.h: This graph shows which files directly or indirectly include this file:

Classes

- class NGLDMFeatures< T, R >

11.39 NGLDMFeatures2DAVG.h File Reference

`#include "NGLDMFeatures2DMRG.h"`
Include dependency graph for NGLDMFeatures2DAVG.h: This graph shows which files directly or indirectly include this file:

Classes

- class NGLDMFeatures2DAVG< T, R >

11.40 NGLDMFeatures2DMRG.h File Reference

`#include "GLRLMFeatures.h"`
Include dependency graph for NGLDMFeatures2DMRG.h: This graph shows which files directly or indirectly include this file:

Classes

- class NGLDMFeatures2DMRG< T, R >

11.41 NGLDMFeatures2DWOMerge.h File Reference

`#include "NGLDMFeatures.h"`
Include dependency graph for NGLDMFeatures2DWOMerge.h:

Classes

- class NGLDMFeatures2DWOMerge< T, R >

11.42 NGLDMFeatures3D.h File Reference

```
#include "NGLDMFeatures2DMRG.h"
```
Include dependency graph for NGLDMFeatures3D.h: This graph shows which files directly or indirectly include this file:

Classes

- class NGLDMFeatures3D< T, R >

11.43 NGTDM.h File Reference

```
#include <iostream>
#include "boost/multi_array.hpp"
#include "image.h"
```
Include dependency graph for NGTDM.h: This graph shows which files directly or indirectly include this file:

Classes

- class NGTDMFeatures< T, R >

11.44 NGTDM2DAVG.h File Reference

```
#include "NGTDM2DMRG.h"
```
Include dependency graph for NGTDM2DAVG.h: This graph shows which files directly or indirectly include this file:

Classes

- class NGTDM2DAVG< T, R >

11.45 NGTDM2DMRG.h File Reference

```
#include <iostream>
#include "boost/multi_array.hpp"
#include "image.h"
```
Include dependency graph for NGTDM2DMRG.h: This graph shows which files directly or indirectly include this file:

Classes

- class NGTDMFeatures2DMRG< T, R >

11.46 NGTDM2DWOMerge.h File Reference

```
#include "NGTDM.h"
```
Include dependency graph for NGTDM2DWOMerge.h:

Classes

- class NGTDM2DWOMerge< T, R >

11.47 NGTDM3D.h File Reference

```
#include "NGTDM2DMRG.h"
```
Include dependency graph for NGTDM3D.h: This graph shows which files directly or indirectly include this file:

Classes

- class NGTDMFeatures3D< T, R >

11.48 readConfigFile.h File Reference

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/ini_parser.hpp>
#include <sys/types.h>
#include <sys/stat.h>
#include "string"
#include "featureCalculation.h"
```
Include dependency graph for readConfigFile.h: This graph shows which files directly or indirectly include this file:

Classes

- class ConfigFile

Typedefs

- typedef boost::property_tree::ptree **config**

11.49 readImages.h File Reference

```
#include "readInFeatureSelection.h"
#include "morphologicalFeatures.h"
#include "itkTypes.h"
#include <fstream>
#include <iterator>
#include <algorithm>
#include <boost/lexical_cast.hpp>
#include <bitset>
#include <sstream>
#include "readImages.cpp"
```

Include dependency graph for readImages.h: This graph shows which files directly or indirectly include this file:

Functions

- ImageType::Pointer readImage (string imageName)
- RegionType **getBoundingBoxMask** (ImageType ∗mask)
- ImageType::Pointer **getImageMasked** (ImageType ∗image, RegionType boundingBoxRegion)
- ImageType::Pointer **getMaskNewSpacing** (ImageType ∗imageFiltered, ImageType ∗maskFiltered)
- ImageType::Pointer **smoothImage** (ImageType ∗image, float kernel)
- vector< int > **getImageSizeInterpolated** (ImageType ∗imageFiltered, ImageType::SizeType imageSize, double(&outputSpacing)[3], ConfigFile config)
- ImageType::Pointer **maskValues2One** (ImageType ∗originalMask)

11.49.1 Function Documentation

11.49.1.1 readImage()

```
ImageType::Pointer readImage (
            string imageName )
```

The method readImage reads the itk-image

Parameters

in		

11.50 statisticalFeatures.h File Reference

```
#include "functional"
#include <iostream>
#include "boost/multi_array.hpp"
#include "math.h"
#include "matrixFunctions.h"
```

```
#include "helpFunctions.h"
#include "vectorFunctions.h"
#include <typeinfo>
#include <fstream>
#include <string>
#include <boost/bind.hpp>
#include <boost/accumulators/accumulators.hpp>
#include <boost/accumulators/statistics/stats.hpp>
#include <boost/accumulators/statistics/mean.hpp>
#include <boost/accumulators/statistics/moment.hpp>
#include <boost/accumulators/statistics/median.hpp>
#include <boost/accumulators/statistics/kurtosis.hpp>
#include <boost/accumulators/statistics/variance.hpp>
#include <boost/accumulators/statistics/skewness.hpp>
#include <boost/accumulators/statistics/p_square_quantile.hpp>
#include <boost/accumulators/statistics/extended_p_square_quantile.hpp>
#include <boost/iterator/filter_iterator.hpp>
#include <boost/cstdlib.hpp>
```

Include dependency graph for statisticalFeatures.h: This graph shows which files directly or indirectly include this file:

Classes

- class StatisticalFeatures< T, R >

Index