DOCUMENTATION

Matlab toolbox for pet / ct image segmentation with the Chan-Vese model

Westfälische
Wilhelms-Universität
Münster

Institute for Numerical and Applied Mathematics
University of Muenster • Germany
a.gripshoefer@gmx.de • s_west05@math.uni-muenster.de

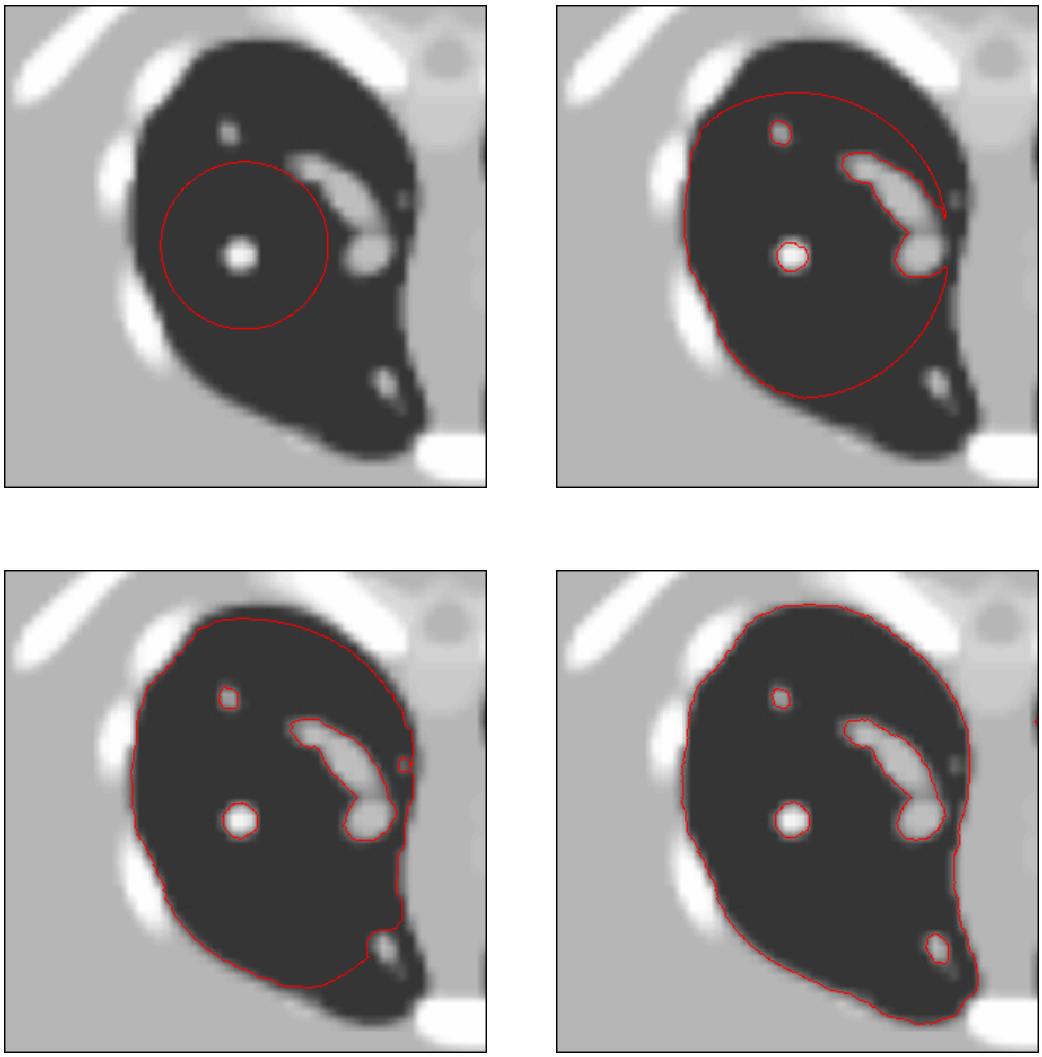
1. Introduction into the toolbox

The Matlab toolbox for pet / ct image segmentation with the Chan-Vese model was developed during the practical course "Medicine and Mathematics" in summer semester 2008 at the University of Muenster, Germany.

It was intended to be used for the purpose of determining a segmentation of certain structures in ct data, which is needed to correct the partial volume effect in pet imaging. As no special characteristics of ct / pet data are used it can be useful in any other 2D / 3D segmentation tasks, too.

With the toolbox data can be imported, visualized and prepared for segmentation. Besides that it offers opportunities for segmenting 2D images and 3D volume data and visualizing the results.

Example in 2D:



Toolbox installation:

1. Unpack the toolbox ('chanvese_tb_1.0.zip') to a directory of your choice
2. Optionally unpack the examples folder ('examples.zip') to a directory of your choice
3. Start Matlab
4. Add directory containing the toolbox to the Matlab path.
   To do this:
   a) choose toolbox directory as Matlab working directory,
      execute the following commands:

```
addpath(cd); savepath;
```

   b) or alternatively use the Matlab GUI to do this
      ("File" --> "Set Path..." --> "Add Folder...")

Now the toolbox functions are accessible from an arbitrary working directory.

2. Examples for segmentation (see additional examples folder)

Segmentation of a 2D image:

First the image ('examples/2D/ncat10mm/ct_ncat_10mm.jpg') which should be segmented has to be imported as a greyscale image. This can be done by the Matlab commands `imread` and `rgb2gray` (change your working directory to the directory above first):

```
u0 = double(rgb2gray(imread('ct_ncat_10mm.jpg')));
```

Then you have to create an initial contour `phi`. This can be done via the toolbox function `createPhi2D`, which creates `phi` as the maximum of different circle functions. For example 8 circles in horizontal and 7 circles in vertical direction with radius 15:

```
phi = createPhi2D(u0, 8, 7, 15, true);
```

Use `ChanVese2D` for segmentation:

```
pause off;
ChanVese2D(u0, phi, 1, 1, 0.01, 20);
```

Segmentation of 3D volume data:

First of all the ct and pet data are imported by `readData` and the data values are scaled on the interval [0,1]:

```
[ctdata, mmPerPixel_xyz_ct] = readData('CT.hv', true);
[petdata, mmPerPixel_xyz_pet] = readData('PET.hv', true);
```

(The header files 'CT.hv' and 'PET.hv' of the artificial example data sets are located in 'examples/3D/ncat10mm'. Change your working directory to there first or include path in filenames…)

If – as usual in pet / ct imaging – the two data sets are of different size, you can match them via `matchData` (not necessary in case of the artificial example data sets):

```
petdata = matchData(ctdata, mmPerPixel_xyz_ct, petdata, mmPerPixel_xyz_pet);
```

The pet data were imported to locate a region of interest (ROI) containing a tumour, which we want to segment in the ct data afterwards. Determine such a ROI in the pet data via `getROI3D`:

```
[x1, x2] = getROI3D(petdata, 0.05);
```

To view the two data sets and the ROI (and later on the segmentation) in the toolbox viewer `xyzViewData`, you have to pass the viewer a Matlab struct with the following fields:

```
                ctdata:  [3D array containing the ct data]
               petdata:  [3D array containing the pet data]
                 ROIx1:  [coordinates of the first corner of the ROI]
                 ROIx2:  [coordinates of the second corner of the ROI]
segmentationVertices:  [array of 3D segmentation vertices]
```

If one or more of the above fields are not available in the struct the viewer does not give you the according display opportunity. For example such struct (let us call it `data`) is created by:

```
data.ctdata = ctdata;
data.petdata = petdata;
[data.ROIx1, data.ROIx2] = getROI3D(petdata, 0.05);
```

In this case it looks like:

```
 ctdata: [128x128x128 double]
petdata: [128x128x128 double]
  ROIx1: [60 33 53]
  ROIx2: [69 47 67]
```

Only to handle the ROI and to segment the data inside of this region, you have to cut out the region (plus boundary) out of the ct and pet data and create a second struct (let us call it `data2`):

```
[cuttedROI, cuttedROIpet] = cutROI3D(data.ctdata, data.petdata, 0.05, 3);
data2.ctdata = cuttedROI;
data2.petdata = cuttedROIpet;
```

or a little shorter:

```
[data2.ctdata, data2.petdata] = cutROI3D(data.ctdata, data.petdata, 0.05, 3);
```

Then create an initial contour `phi` via `createPhi3D`. Like in 2D, `phi` is created as the maximum of different ball functions. E.g. 2 balls in each direction with radius 3 are created by:

```
phi = createPhi3D(data2.ctdata, 2, 2, 2, 3, true);
```

Use `ChanVese3D` for segmentation:

```
segmentation = ChanVese3D(data2.ctdata, phi, 255^2, 255^2, 0.001, 50);
```

Extract contour points from the result `segmentation` via `extractSegmentationVertices3D` and put the extracted list of segmentation vertices into `data2`:

```
data2.segmentationVertices = extractSegmentationVertices3D(segmentation);
```

Now `data2` looks like:

```
                ctdata: [16x21x21 double]
               petdata: [16x21x21 double]
segmentationVertices: [640x3 double]
```

By passing `data2` to the viewer `xyzViewData` you can see the ct and pet data and the segmentation in the viewer's GUI:

```
xyzViewData(data2);
```

3. Documentation of the m-files

The toolbox consists of the following m-files:

ChanVese2D.m
ChanVese2D_alternative.m
ChanVese3D.m
createPhi2D.m
createPhi3D.m
cutROI3D.m
drawSegmentation.m
exportSegmentationASCII.m
extractSegmentationVertices3D.m
getROI3D.m
matchData.m
petctFusion.m
readData.m
scaleData.m
trilinearInterpol.m
xyzViewData.m
zViewData.m

The following descriptions of each function can also be read in Matlab by the command:

```
help <function name>
```

ChanVese2D:

CHANVESE2D Segmentation of an image into two regions via the Chan-Vese-Model

phi = ChanVese2D(u0, phi, lam1, lam0, nue, iterationCount, h, dt)

Segmentation of the image u0 into two regions via the Chan-Vese active contour model. An initial contour has to be given implicitly as the 0-isocontour of a discrete level set function phi and the final contour segmenting the image into two regions is returned in the same way.

As usual in the level set framework u0 and phi must have the same size (as both are discrete functions on the same domain) and the two regions consist of those points on the domain where phi > 0 and those where phi < 0, respectively (the 0-isocontour is the interface between both regions).

lam1, lam0 weight the data values of the regions:
lam1: region inside the (0-iso-)contour
lam0: region outside the (0-iso-)contour

nue weights the length of the (0-iso-)contour:
the bigger the nue, the stronger the minimization of the length

iterationCount defines the number of iterations

h and dt are optional parameters of the discretization:
h is the space step (default: h = 1)
dt is the time step (default: dt = 0.1)

ChanVese2D_alternative:

CHANVESE2D_ALTERNATIVE Segmentation of an image into two regions via the Chan-Vese-Model

phi = ChanVese2D_alternative(u0, phi, lam1, lam0, nue, iterationCount, h, dt)

Segmentation of the image u0 into two regions via the Chan-Vese active contour model. An initial contour has to be given implicitly as the 0-isocontour of a discrete level set function phi and the final contour segmenting the image into two regions is returned in the same way.

As usual in the level set framework u0 and phi must have the same size (as both are discrete functions on the same domain) and the two regions consist of those points on the domain where phi > 0 and those where phi < 0, respectively (the 0-isocontour is the interface between both regions).

In difference to the function ChanVese2D the discretization of the following paper is used:

Vese, L. and Chan, T. 2002. A multiphase level set framework for image segmentation using the Mumford and Shah Model

lam1, lam0 weight the data values of the regions:
lam1: region inside the (0-iso-)contour
lam0: region outside the (0-iso-)contour

nue weights the length of the (0-iso-)contour:
the bigger the nue, the stronger the minimization of the length

iterationCount defines the number of iterations

h and dt are optional parameters of the discretization:
h is the space step (default: h = 1)
dt is the time step (default: dt = 0.1)

ChanVese3D:

CHANVESE3D Segmentation of 3D volumedata into two regions via the Chan-Vese-Model

phi = ChanVese3D(u0, phi, lam1, lam0, nue, iterationCount, h, dt)

Segmentation of 3D volumedata u0 into two regions via the Chan-Vese active contour model. An initial surface has to be given implicitly as the 0-isosurface of a discrete level set function phi and the final surface segmenting the volumedata into two regions is returned in the same way.

As usual in the level set framework u0 and phi must have the same size (as both are discrete functions on the same domain) and the two regions consist of those points on the domain where phi > 0 and those where phi < 0, respectively (the 0-isosurface is the interface between both regions).

lam1, lam0 weight the data values of the regions:
lam1: region inside the (0-iso-)surface
lam0: region outside the (0-iso-)surface

nue weights the area of the (0-iso-)surface:
the bigger the nue, the stronger the minimization of the area

iterationCount defines the number of iterations

h and dt are optional parameters of the discretization:
h is the space step (default: h = 1)
dt is the time step (default: dt = 0.1)

createPhi2D:

CREATEPHI2D Creates an initial contour consisting of several circles

phi = createPhi2D(u0, a, b, r, visualizePhi)

Creates an initial contour as the 0-isocontour of a discrete level set function phi consisting of a circles in horizontal and b circles in vertical direction, each with radius r. In detail phi is computed as a signed distance function.

u0 is the image which is being segmented

visualizePhi is an optional parameter:
defines whether the initial contour is visualized in the image or not (default: visualizePhi = false)

createPhi3D:

CREATEPHI3D Creates an initial surface consisting of several balls

phi = createPhi3D(u0, a, b, c, r, visualizePhi)

Creates an initial surface as the 0-isosurface of a discrete level set function phi consisting of a balls in horizontal, b balls in vertical direction and c balls in the depth, each with radius r. In detail phi is computed as a signed distance function.

u0 are the 3D volumedata which are being segmented

visualizePhi is an optional parameter:
defines whether the initial surface is visualized or not (default: visualizePhi = false)

cutROI3D:

CUTROI3D Cuts a region of interest out of ct (and pet) data

[cuttedROI, cuttedROIpet] = cutROI3D(ct, pet, epsilon, boundary)

Cuts a region of interest (ROI) out of ct data with a given boundary. This ROI is determined from the pet data using getROI3D. Optionally the corresponding section is cutted out of the pet data, too.

epsilon: threshold for the tolerance in ROI detection

boundary is an optional parameter:
defines the size of the additional boundary added to the ROI in each direction, measured in pixels (default: boundary = 3)

[ROIct] = cutROI3D(...) returns ROI in ct data
[ROIct, ROIpet] = cutROI3D(...) returns ROI in ct data and pet data

drawSegmentation:

DRAWSEGMENTATION Draws the segmentation into the image u0

drawSegmentation(u0, phi, writeToDisk, picNumber)

Draws the segmentation, which is represented by the 0-isocontour of the discrete level set function phi, into the image u0.

writeToDisk and picNumber are optional parameters:
writeToDisk decides whether the image will be saved to the file 'pic<picNumber>.png' or not (default: writeToDisk = false)

exportSegmentationASCII:

EXPORTSEGMENTATIONASCII Exports vertices of the 3D segmentation into an ASCII file

exportSegmentationASCII(vertices3D, filename)

Exports the array of 3D segmentation vertices (vertices3D) into an ASCII file (filename). Each line of this ASCII file will contain the coordinates x, y and z of one vertex, seperated by a tab stop.

extractSegmentationVertices3D:

EXTRACTSEGMENTATIONVERTICES3D Extracts an array of 3D segmentation vertices

vertices = extractSegmentationVertices3D(segmentation)

Extracts an array of 3D segmentation vertices from the level set function (segmentation). Each row of the array will contain the coordinates x, y and z of one vertex.

getROI3d:

GETROI3D Detects a region of interest in 3D pet data

[x1,x2] = getROI3D(u0, epsilon)

Detects a region of interest (ROI) in 3D pet data (u0). It is
cuboid-shaped and is returned via two coordinates x1 and x2
which are spanning up the cuboid:
x1 = coordinates (y,x,z) of the upper-left frontmost corner of the ROI
x2 = coordinates (y,x,z) of the lower-right backmost corner of the ROI

Controlled by epsilon the algorithm expands a cuboid in each direction
periodicly, starting with the pixel which has the maximum value in u0.

epsilon: threshold for the tolerance in ROI detection

This function is intended to be used by cutROI3D...
For real ROI extraction use cutROI3D.

matchData:

MATCHDATA Matches petdata with ctdata via trilinear interpolation

petdata = matchData(ctdata, mmPP_xyz_ct, petdata, mmPP_xyz_pet)

Matches 3D petdata with 3D ctdata via trilinear interpolation.

mmPP_xyz_ct: property of the 3D ctdata; vector containing the millimetres per pixel in x, y and z direction
mmPP_xyz_pet: property of the 3D petdata; vector containing the millimetres per pixel in x, y and z direction

petctFusion:

PETCTFUSION Creates a petct image out of a ct and pet image

[petct,rgbct,rgbpet] = petctFusion(ct, pet, visualize)

Creates a petct image out of a ct and pet image, which have to be
matched to each other. For this fusion suitable ct and pet colormaps
are used.
Optionally the fused petct, ct and pet images are visualized.

visualize is an optional parameter:
decides whether the fused petct, ct and pet images should be visualized, or not (default: visualize = false);

[petct] = petctFusion(...) returns fused petct image
[petct,ct] = petctFusion(...) returns fused petct image and the ct image, embedded into rgb color space with a suitable ct colormap
[petct,rgbct,rgbpet] = petctFusion(...) returns fused petct image and the ct and pet images, embedded into rgb color space with suitable ct / pet colormaps

readData:

READDATA Reads 3D ct / pet data out of interfiles

[data, mmPerPixel_xyz] = readData(hvFile, scaled)

Reads 3D ct / pet data out of interfiles. A dataset in the interfile
format is given by a header (hvFile) and a data file. The header file
contains the necessary information about the data file.
Currently 'float' and 'signed integer' are supported data types for
data files. Both, little endian and big endian byte ordering is supported.
readData returns a 3D Array containing the data; the ordering of the
dimensions is (Y,X,Z).
Optionally the data values can be scaled to the interval [0,1].

scaled is an optional parameter:
decides whether data values are scaled to the interval [0,1], or
not (default: scaled = true);

[data] = readData(...) returns data
[data, mmPP] = readData(...) returns data and pixel measurements

scaleData:

SCALEDATA Scales data values of 2D images / 3D volume data

scaledData = scaleData(data, newMax, newMin, maximum, minimum)

Scales data values of ct / pet images and volume data linear to the
interval [newMin,newMax]. The interval [0,1] is chosen by default.
Alternatively scaleData can be used for contrast stretching by scaling
only a small interval of interesting data values.

newMax and newMin are optional parameters:
scaleData(data) scales data values to the interval [0,1]
scaleData(data, newMax, newMin) scales data values to the interval [newMin,newMax]

maximum and minimum are optional parameters for contrast stretching:
scaleData(data, newMax, newMin, maximum, minimum) scales data values from
the interval [minimum,maximum] to the interval [newMin,newMax];
data values lower than minimum or higher than maximum are cutted
off

trilinearInterpol:

TRILINEARINTERPOL Scales 3D ct / pet data via trilinear interpolation

g = trilinearInterpol(f, s_x, s_y, s_z)

Scales the 3D ct / pet data f via trilinear interpolation. In each
direction x,y,z the scaling factors s_x, s_y and s_z are used.

This function is intended to be used by matchData...

xyzViewData:

XYZVIEWDATA "xyzView - Data Explorer" for 3D ct and pet data, ROI and segmentation visualization

xyzViewData(data)

Visualizes 3D ct and pet data by displaying the axial, coronal and
sagittal planes and offers the possibility of displaying a region of
interest (ROI) and a 3D segmentation of the data.

There are two different modes of operation: simple and extended mode.

In simple operation mode the input argument data is a 3D array of
arbitrary volume data, with data values scaled to the interval [0,1].
Depending on the users choice in the GUI it is visualized either in ct
modality or in pet modality.

In the more interesting extended mode the input argument data is a
struct containing the following fields

ctdata:	[3D array containing the ct data]
petdata:	[3D array containing the pet data]
ROIx1:	[coordinates of the first corner of the ROI]
ROIx2:	[coordinates of the second corner of the ROI]
segmentationVertices:	[array of 3D segmentation vertices]

The ctdata and petdata data values must be scaled to the interval [0,1]
the ordering of the dimensions of data and ROI coordinates must be
(Y,X,Z) and ct data and pet data must be matched to each other.
The visualization then is configured depending on which of the above
fields are available (at least one type of data must be available) and
the users choice in the GUI.

zViewData:

ZVIEWDATA zViewData data explorer for fast 3D volume data visualization

zViewData(data)

Visualizes arbitrary 3D volumedata data by displaying the axial plane
in ct modality. The data are given as a 3D array with data values scaled
to the interval [0,1] and dimension ordering (Y,X,Z).

4. References

Chan, T. and Vese, L. 2001. Active contours without edges. IEEE-IP, 10(2):266-277

Vese, L. and Chan, T. 2002. A multiphase level set framework for image segmentation using the Mumford and Shah model. IJCV, 50(3):271-293

For future releases and additional information see also at:

http://wwwmath.uni-muenster.de/num/Vorlesungen/MedizinUndMathematik/