Function ms_PCNN3D

Purpose: performing brain extraction on rodent data (anatomical and/or EPIs)

Usage: `[ Pout, finalBrainSize ] = ms_PCNN3D( P, BrSize )`
P: filename of nifty image
BrSize: limits of assumed final brain size (e.g. `BrSize=[350,650]*1000;`)
Pout: filename of extracted brain (suffix '_brain')
finalBrainSize: volume in mm³ of the estimated brain extraction

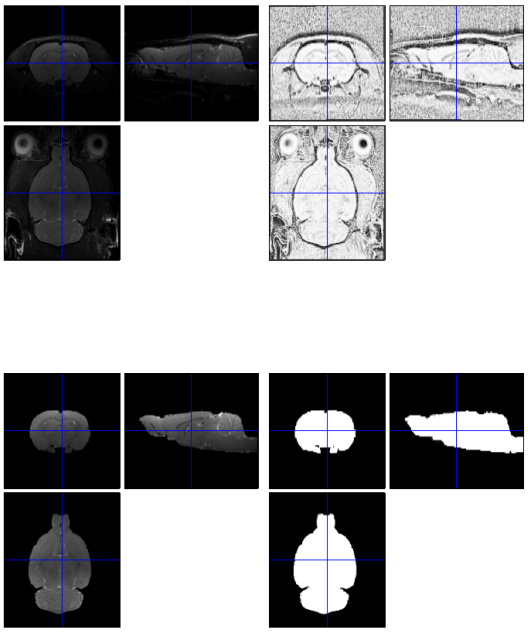
```
verb = 0; % verbose mode
shows a figure with images of the PCNN variables during the iterations;
nice to see if you want to understand it but in general not very useful

ToSkip = 0; % iterations to skip the actual brain mask calculation; but
PCNN runs!
Not recommended anymore

showResult = 0; % final check_reg and iteration figure
shows the final results in spm_check_reg and a figure of "volume evolution"
```

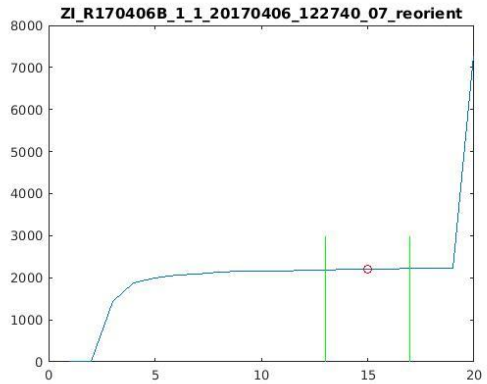
What you should see

If you set showResult=1 you will see a spm_check_registration and a figure like that:



Starting from left top:

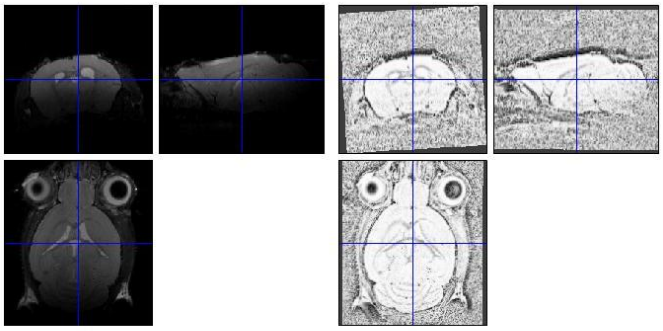
Original image
Preprocessed image
Brain extraction
Mask

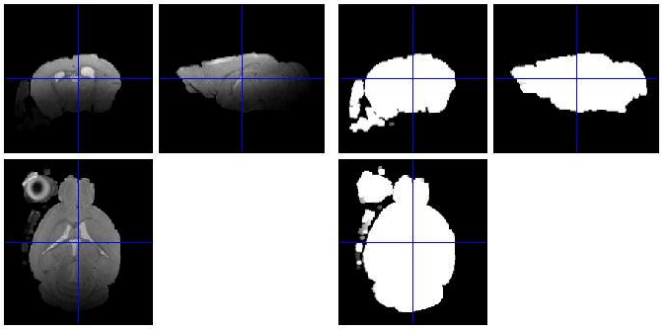


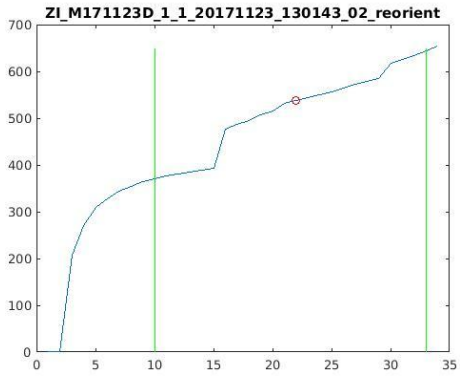
Additionally, on the right the "volume evolution". In this example everything worked fine. You want to see this plateau in which the brain volume only changes slightly which indicates that it worked fine. There is an algorithm which estimates the best iteration based on the BrSize variable and this evolution. It doesn't work perfectly and a big, flat plateau helps! The green lines indicate the assumed "best range" of iterations.

What if it doesn't work fine?

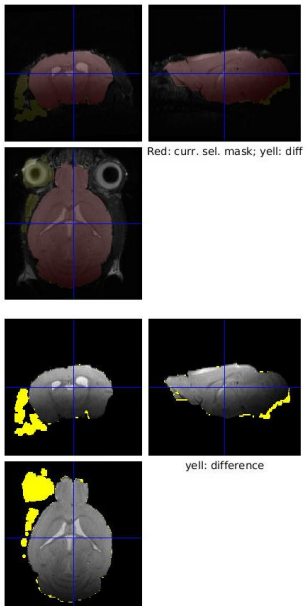
The example above is of good data quality. There is an obvious bias field in the original image but the image preprocessing did a good job to get rid of it. But that is not always the case:

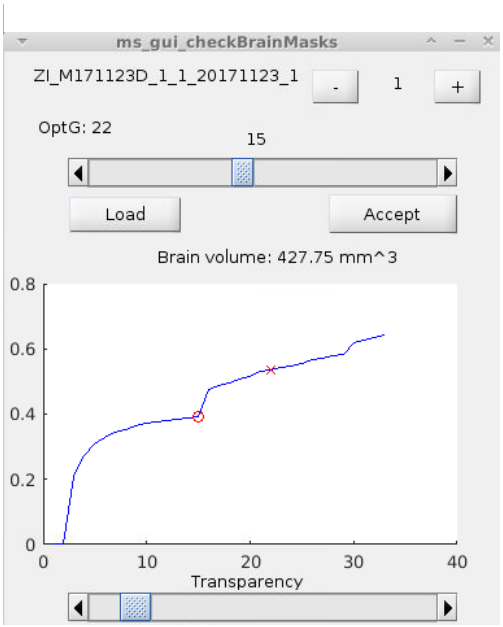






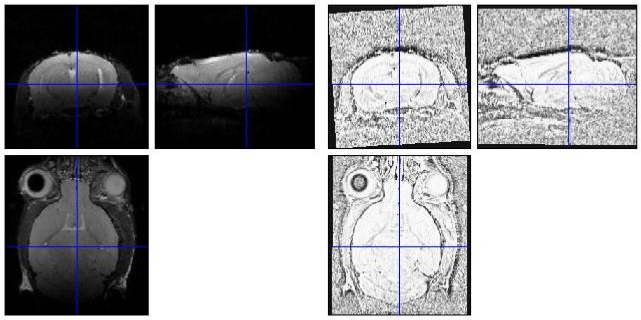
This data has a very strong bias field and the brain vanishes into noise. The voxel evolution has no plateau and the found optimal iteration is a joke. In this case you can use ms_gui_checkBrainMasks

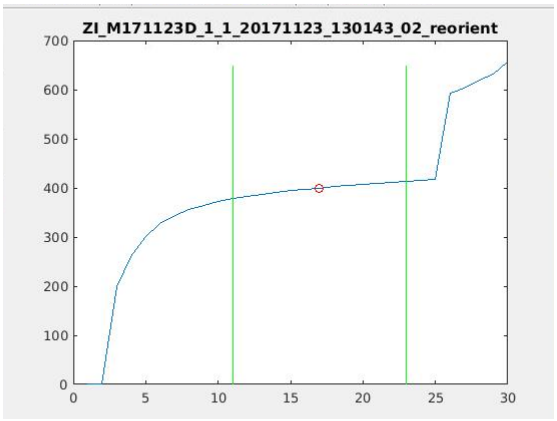


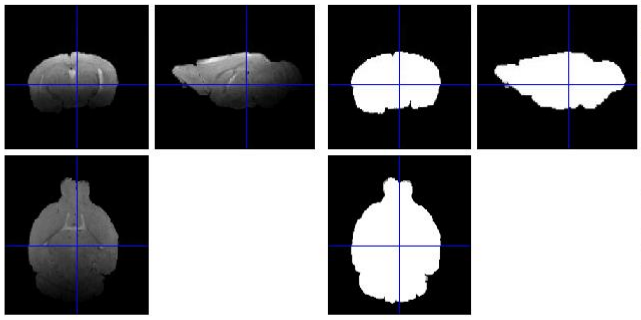


The cross marks the initially found optimal iteration. The circle is the currently selected mask. The difference of both is indicated in yellow. You can accept this mask and the fact that a big part of the cerebellum is noise anyway or:

1. Try to perform a bias correction and run it again
2. Take the processed image and run the PCNN again to maybe get at least a better mask
3. Tweak the image preprocessing (ms_preprocImage). There, changing the value of the imgaussfilt3 function or increasing the value of gradient subtraction can help
4. **Maybe the best point to start**: Change the parameter for morphological smoothing **p**. In the bad example above it was set to 4. Since there are "regions" outside the brain included it is very likely that there is a "little bridge" connecting the brain and outer brain regions. When we change p to 5 we get this:







By higher smoothing we were able to remove the bridge and isolate the brain for more iterations. p=6 would also be an option.

How it works

The algorithm is based on a pulse-coupled neural network (**PCNN**; an iterative process in which neurons (the image voxel) are activated based on several parameters. There are Linking and Feeding layers but the most interesting part is the firing of neurons which happens when their potential (U) is higher than a threshold (T). This thing is written in 6 lines

```
K = convn(Y,M,'same');
Fn = exp(-log(2)/0.3).*F + 0.01*K + S;
Ln = exp(-log(2)/1).*L + 0.2.*K;
Un = Fn.*(1+0.2*Ln);
Tn = exp(-log(2)/10).*T + 20.*Y;
Yn = double(Un>Tn);
```

The starting values are all set to zero except T which is set to one. As you can see the firing of a neuron (Y) depends on U>T. The potential U of a neuron is influenced by feeding F and linking L and grows while its threshold T decreases slightly for every iteration. To prevent a neuron from firing again immediately its T is set to a high number.
This leads to the firing of different neurons (over the iterations) which is influenced by F and L and thus also by its initial value of the image. High image values will fire first, then influence their neighbors (that's why there is K) to maybe also fire (but that also depends on their image value).

How can the PCNN be used for brain extraction?

Since Y alone is interesting to look at but quite useless in general, the main idea is to 'collect' all the neurons which fired in a variable A.

```
A = A | logical(Y);
```

A is the basis for all further processing. Since A alone is almost as useless as Y, you make the assumption, that the biggest region in your image should be the brain at some point of the iteration process. That's a reasonable assumption since we measure the brain. So:

- The input shall be a usual anatomical image. There is the brain but also eyes, surrounding tissue, and background noise.
- After every iteration we search/find the regions in A (which means connected voxels) and pick the biggest one as 'brain'.
- Since the eyes are typically very bright the first neurons will fire there at the beginning of the process. They will influence their neighbors, their potential U increases while their T decreases and so on.. at this point the biggest region is one of the eyes.
- At some point a neuron in the brain will fire (since T is steadily decreasing and U increasing that has to happen).
- Like before in the eyes, this neuron will influence its neighbors which will fire if they are "similar" (which means if they have a similar brightness).
- After a few iterations the biggest region is no longer an eye but the brain. More and more neurons within the brain will fire and being "collected" in A. It is like a 'wave' of newly activated neurons traveling through the brain.
- At some point, newly activated neurons are close to the border of the brain and start to influence voxel there. But since the border is of very low brightness (image value) the firing of these neurons is delayed and the wave can reach every part of the brain.
- This 'behavior' can be seen in the plateau mentioned above.
- After some further iterations, even the border neurons are activated and in the best case the brain region connects immediately with big regions outside the brain (remember: outside the brain are also activated regions) leading to that the biggest region exceeds the assumed maximum brain size and the iteration stops.

That's it.. this is the way it works. But there are so many things more:

1. It happens of course quite often that 'border neurons' are activated too early. Therefore, A is morphologically smoothed (erosion, dilation) to erase small bridges connecting the brain with outer regions. See `calcBrainMask(A,se)`. The exact way how you do it has an influence on the outcome. What works for high res images isn't necessarily working well for EPIs too. For anatomical images a p (morph. Smoothing parameter) of ~5 works well. For low res EPIs better take sth around 2.
2. You want the voxels in your brain as similar as possible and the nearby surrounding as different as possible. That doesn't always work, but a bias-correction for example helps. That's why there is a pseudo bias-correction and a gradient subtraction implemented in the image preprocessing.

3. Finding the correct optimal iteration is sometimes difficult. There are several approaches to do so implemented. Best results gives so far the “own approach”. But sometimes it fails. Then the original “paper approach” is used. All approaches state their results at the end of the iteration. Have a look there.

Feel free to play around with all the parameters. Sometimes it’s enough to set the assumed brain size closer to the real one. Sometimes you have to dig a bit deeper ;)